EN MARKT & TECHNIK BOG

HORST-RAINER HENNING

AMIGA

LÆRE- & HÅNDBOG

BASIC-programmering

Lær programmering — den lette vej!

DANSK/NORSK udgave

inklusiv diskette

Computerbogs-forlag: NCS, box 105, DK-6950 Ringkøbing

ASSEMBLER

AMIGA PROFIMAT

Den komplette programpakke for enhver Amiga Maskinsprogsprogrammør.
Extrem hurtig, er skrevet i Assembler.

- **Integreret Editor, Debugger, Disassembler og Reassembler.**
- **Omfangsrigt System-bibliotek.**
- **Kører under CLI & Workbench.**
- **Relativ og absolut Code.**
- **Fejlsøgningsfunktioner.**
- **Cross-Reference-Liste.**
- **Fuld 32 Bits aritmetik.**
- **Debugger med 68020-Singel-Step-Emulation.**
- **Kører på Amiga med 512 Kb og Kickstart 1.2.**

Kun kr. 498,00
inkl. moms

NORDIC COMPUTER SOFTWARE
SMEDEGADE 7
POSTBOX 105 . DK-6950 RINGKØBING

AMIGA BASIC

EN MARKT & TECHNIK BOG

HORST-RAINER HENNING

AMIGA

LÆRE- & HÅNDBOG

BASIC-programmering

Lær programmering — den lette vej!

DANSK/NORSK udgave

NORDIC COMPUTER SOFTWARE
SMEDEGADE 7 . BOX 105 . DK-6950 RINGKØBING
1989

KUNDESERVICE:
TLF. 07 32 02 02, KL. 09.00-10.00

Org.-titel: Programmiren mit
AMIGA BASIC
af
Horst-Rainer Henning
1987 by Markt & Technik Verlag Aktiengesellschaft
8013 Haar bei MÜnchen, Vesttyskland

Copyright 1989 Nordic Computer Software
Smedegade 7
DK-6950 Ringkøbing

Tryk: Tarm Bogtryk & Offset A/S

Dansk oversættelse: E.J. Fjerdingstad

ISBN 87-7283-030-1

Alle rettigheder til den danske version tilhører Nordic Computer Software. Dette værk må under ingen form (fotokopi, aftryk el. lign) reproduceres uden skriftlig tilladelse fra udgiveren. Ligeledes må dette værk , eller dele heraf, ikke udbredes via elektroniske medier.
Programmer, programeksempler mv. er omfattet af lov om copyright. Disse må kun anvendes til personlige- eller undervisningsformål og må ikke anvendes kommercielt.

Vigtigt

Alle koblinger, tekniske angivelser og programmer er udarbejdet eller sammenstillet af forfatteren med den største omhyggelighed og produceret under iagtagelse af effektive kontrolforanstaltninger. Alligevel kan fejl ikke udelukkes. N.C.S ser sig derfor nødt til at bemærke, at det ikke kan påtage sig sig nogen garanti eller juridisk ansvar eller anden hæftelse for følger af fejlagtige oplysninger. Forlaget er til enhver tid taknemlig for skriftlige oplysninger om eventuelle fejl.

INDHOLDSFORTEGNELSE

FORORD

1 AMIGA-BASICS GRUNDLAG

1.1 Det grundlæggende om Amigas betjening 8
1.2 Amigas funktionsmåder 8
1.2.1 Loadning af Amiga-Basic 10
1.2.2 Den direkte funktionsmåde 11
1.2.3 Program-funktionsmåden 12
1.2.4 Læsning og editering af programmer 12
1.2.5 Styring af programudførelsen 15
1.3 Loadning og gemning af programmer 17
1.3.1 En rute fører gennem filerne 17
1.3.2 New 18
1.3.3 Loadning af et program 19
1.3.4 Save 20
1.3.5 Save As 20
1.3.6 Quit 22
1.3.7 Ombenævnelse og sletning af programmer 22
1.4 Styring af programforløbet 22
1.4.1 Labels 23
1.4.2 GOTO, GOSUB 23
1.4.3 FOR NEXT-løkker 25
1.4.4 IF-sætningen 27
1.4.5 WHILE-WEND løkken 29
1.4.6 Hvem vækker Tornerose? 30
1.5 Variabler 32
1.5.1 Barnet skal have et navn 32
1.5.2 Variabeltyper 33
1.5.3 En specialitet - feltvariabler 37
1.6 Basic 1*1 39
1.6.1 Over stikker under 39
1.6.2 Logiske operatorer 41
1.6.3 Sammenligningsoperatorer 42

2 GRAFIK

2.1 Den første grafik-kommando 44
2.2 Åbning og lukning af vinduer 46
2.3 Hvorhen med programmerne? 51
2.4 Mere farve 54
2.5 Regnbue 55

2.6 Dam 59
2.7 Tordenvejr 64
2.8 PAINTAmiga 69
2.8.1 PAINTAmiga i detaljer 71
2.8.2 PAINTAmiga kører 75
2.8.3 PAINTAmiga-håndbog 77

3 FILER

3.1 Printning af tekst 80
3.2 Hardcopy 81
3.3 IFF, chunks og den slags 86
3.3.1 Håndtering af hukommelsesadresser 87
3.3.2 Indlæsning af IFF-grafik 89
3.3.3. Gemning i INTERCHANGEABLE FILE FORMAT 95
3.4 Sekventielle filer 97
3.5 Kogebog 100
3.6 Talsalat 104
3.7 Den der arbejder, gør også fejl 107
3.8 Mine biler 109
3.9 Direkte tilgang - hurtig og alsidig 113
3.10 Terminskalender 117

4 SPRITES, BOBS OG LYD

4.1 Hallo - det er Amiga 120
4.2 Det hele går bedre med lidt musik 124
4.3 Toner - blide bølger eller kantede 126
4.4 Crash, zoom & wham - lydeffekter 130
4.5 Levende billeder - sprites og BOBs 134
4.6 Kunstflyvning 138
4.6.1 Joystick'et 144
4.7 Nedskydning 146
4.8 Så ruller skærmen 151

5 TIPS OG TRICKS

5.1 Disk uden diskette 158
5.2 Stor, større, størst 159
5.3 Specialtegn 160
5.3.1 Tegnkoder ved tastaturaflæsning 160
5.3.2 Tegnkoder til PRINT-sætningen 161
5.4 Tilfældet 162
5.5 At stå på hovedet 164
5.6 BOBs og sprites på en anden måde 166

5.7 Automatisk dimensionering af felter 170
5.8 Lidt matematik 171
5.9 Resten 174

6 AFSKED 176

7 OVERSIGT OVER BASIC-KOMMANDOERNE 177

Forord

Efter at have gennemgået mange bøger på engelsk og tysk, er vi kommet til at Basic-bogen skrevet af Horst-Rainer Hennimg er nærmest det flertallet af danske Amiga-ejere ønsker.

I kapitel 4 står en del om programmering af spil, men tro nu ikke at det kun er noget for de læsere, der interesserer sig for spil.

Den indsigt man opnår vedrørende den talende computer(også dansk), computer-musik, lydeffekter, bevægelig billeder o.m.a., kan altsammen udnyttes seriøst til forskellige formål.

Så for Deres egen skyld, spring ikke kapitel 4 over.
Alle i bogen forekommende variabelnavne er markeret med en stjerne foran, for at sikre mod fejltagelser.

Hvor der i bogen fremkommer, f.eks

REM
P 1.5.3-2

er betydningen som følger:

P = program 1 = Kapitel 5 = afsnit 3 = underafsnit -2 = program nr.2 i nævnte underafsnit.

Programmet ligger på den medfølgende diskette.

Mange timers god fornøjelse ønskes af

Forlæggeren

1. AMIGA BASIC'S GRUNDLAG

Af pladsmæssige grunde kan denne bog ikke forklare alle programmeringssproget Basic's begreber i form af en lærebog og samtidig anvendelsen af alle Amigas specielle egenskaber. Hvert kapitel forudsætter derfor en vis grundlæggende viden om sproget Basic og betjeningen af Amigaen.

1.1 Det grundlæggende om Amigas betjening

Før man kan arbejde med Amiga-Basic, må man naturligvis - i det mindste overfladisk - være fortrolig med betjeningen af Amigaen. Har man endnu ikke denne viden, bør man skaffe sig den, ved omhyggelig gennemgang af manualen.

1.2 Amigas funktionsmåder

Nu kan man knytte den første kontakt til Amiga-Basic. Basic-fortolkeren har tre forskellige funktionsmåder.

I den direkte måde bliver Basic-kommandoer udført straks (direkte) efter indtastningen. For at kunne arbejde i program-funktionsmåde, skal man oprette en liste af sætninger. Denne liste kaldes et program. Så snart programmet er startet, befinder man sig i Basic program-funktionsmåde. Den tredje måde kalder man editerings-funktionmåden. Så længe man indtaster og ændrer på programmer, befinder man sig i editerings-funktionsmåden.

For at kunne afprøve ovennævnte, skal man vide, hvordan man kommer ind i Basic.

1.2.1 Loadning af Amiga-Basic

Tag Workbench-disketten ud af det interne diskettedrev, og indlæg disketten Amiga-Extras i stedet for. Åbn så den diskette-ikon, der fremkommer, ved et dobbeltklik med musen. Det nye vindue, viser fire pictogrammer (ikoner): AmigaTutor, affaldsspanden, en skuffe BASICDemos og Amiga-Basic. Basic-fortolkeren "gemmer" sig bag ikonen AmigaBasic. Det drejer sig om et program som ethvert andet (f.eks. textCraft eller DeluxePaint). Basic er nemlig ikke fast indbygget i Amigaen.

Der er tre måder at loade Amiga-Basic på:

- Man indlæser et Basic-program ved at foretage et dobbeltklik på dets ikon. Amiga-Basic loades automatisk med herunder. Det kan man til enhver tid afprøve med et program fra skuffen BASICDemos
-
- Man aktiverer Basic-fortolkeren med et dobbeltklik på ikonen Amiga-Basic, uden samtidig at loade et program.

- Den tredje mulighed giver CLI. Jeg var i tvivl om, hvorvidt jeg skulle medtage denne metode. Hvis man skulle komme til at mangle hukommelse, er det den metode, der bruger mindst hukommelse. Imod den taler, at læseren sikkert ikke vil have nogen større erfaring med CLI. CLI er et særligt interface til operativsystemet og har et stort antal specielle kommandoer. Man er ikke længere i den meget sikre Workbench, der beskytter mod egne uforsigtigheder, men man kan ødelægge hele indholdet af en diskette med en eneste forkert indtastning. Vil man alligevel ikke vil give afkald på muligheden, skal man nøje mærke sig følgende:

Aktiver CLI-ikonen med et dobbeltklik. Det åbner CLI-vinduet. Indtast så:

1df0:Amiga-Basic

(de tegn, der er trykt med fedt, behøver man ikke at indtaste, dem viser CLI på skærmen) og tryk så på RETURN-tasten. Når man senere forlader CLI, lander man igen i dette CLI-vindue. For at lukke vinduet (det har ikke noget lukke-gadget) skal man indtaste denne kommando:

1endcli

og trykke RETURN. CLI-vinduet forsvinder, og man er tilbage i Workbench.

Vælg den sikre vej: gå med musens pil til Amiga-Basic ikonen, og tryk kort to gange på musens valgtast. Diskettedrevet begynder at summe, og kort efter ses to nye vinduer på skærmen; et stort og et der kun er halvt så stort, som dækker halvdelen af det første vindue. Det største af de to vinduer kaldes Output-vinduet, fordi output af de fleste programmer finder sted her. Det mindre vindue (der selvfølgelig som ethvert andet vindue kan forstørres eller formindskes) er List-vinduet. Heri listes senere det netop bearbejdede program, og her indtastes nye programkommandoer eller de allerede tilstedeværende ændres.

1.2.2 Den direkte funktionsmåde

Som man har set i indledningen til Kapitel 1.2, skelnes mellem direkte og program-funktionsmåde i Basic. I den direkte funktionsmåde indtaster man sætninger til Amiga-Basic direkte via tastaturet. På den måde kan man loade og gemme programmer, styre programkørslen og finde fejl. På andre computere kender computeren forskel på de to funktionsmåder ved, at en direkte kommando (som skal udføres med det samme) ikke har noget linienummer.

Da Amiga-Basic ikke behøver sådanne numre, har man valgt en anden fremgangsmåde: de to funktionsmåder arbejder i hver sit vindue. De direkte kommandoer indtastes i Output-vinduet, og alle kommandoer, som skal optages i et program, skal indtastes i List-vinduet.

For at kunne indtaste den første kommando i direkte funktionsmåde, skal man aktivere Output-vinduet ved at klikke med musen midt i vinduet. Cursoren fremkommer så under teksten med Basic-versionen. Med "cursor" menes her den lodrette, orangefarvede streg. Den markerer altid det sted, hvor nyindtastet tekst vil dukke frem. Indtast nu:

```
print "Goddag, jeg er Commodore Amiga"
```

og tryk RETURN. Hvis det bliver nødvendigt at rette en anslagsfejl, gøres det med tasten BACKSPACE (returtasten). Vil man slette hele linien, trykkes tastkombinationen CTRL-C. Nu skal der på skærmen stå:

```
Goddag, jeg er Commodore Amiga
OK
```

Ved hjælp af PRINT har vi givet Amigaen den første kommando i direkte funktionsmåde, og tillige lært en af de mest anvendte Basic-kommandoer at kende

> PRINT liste af udtryk
> viser værdien af liste af udtryk i det aktuelle vindue

I eksemplet har vi givet kommandoen at udlæse en tekst (Goddag, jeg er Commodore Amiga). (Tekster, der skal udlæses, skal altid indtastes i anførelsestegn.) Men man kan lige så godt lave en regneopgave med PRINT. Man behøver blot at skrive en passende formel i stedet for teksten bag ved PRINT. Og hvis man skriver PRINT uden noget bagefter, udlæses en tom linie.

Indtastning i direkte funktionsmåde er ikke begrænset til en enkelt kommando. Man kan indtaste så mange kommandoer, der kan være på en Basic-linie med 255 tegn. De enkelte kommandoer skal så adskilles af et kolon (:).

```
PRINT "Kommandolinie":PRINT 5/9*7:PRINT :PRINT 86-34
```

Tryk RETURN, og det følgende fremkommer:

```
Kommandolinie
3.888889

52
OK
```

Eksemplet viser, at en kommando i direkte funktionsmåde bliver udført straks efter trykket på RETURN-tasten, uden at have været gemt. Resultaterne vises med det samme.

1.2.3 Program-funktionsmåde

Vi kommer nu til den funktionsmåde, man for det meste arbejder i, program-funktionsmåden. Et program er en liste over kommandoer eller sætninger, som Amigaen udfører i den rækkefølge, de blev indtastet, med mindre bestemte kommandoer til styring af forløbet sørger for en anden rækkefølge. Listen kan man gemme, til enhver tid loade igen, ændre og lade udføre (starte). Så snart et program er startet, befinder det sig i program-funktionsmåde, indtil det er afsluttet. Programmøren kan bestemme, at det ikke kan afbrydes. Man kan ikke bruge kommandoer i direkte funktionsmåde, sålænge et program kører.

Før man kan lade et program køre, er man nødt til at skrive det. Hertil skal man aktivere List-vinduet. Hvis det ikke er synligt eller er lukket, kalder man SHOW LIST fra Windows-menuen. Indtast så det følgende program. Det giver nogle linier med tegnet # på skærmen:

```
a$ = "###"
FOR i = 1 TO 20
b$ = b$ + a$
PRINT b$
NEXT
```

Start programmet med menupunktet Start fra Run-menuen. Er der sket en fejl under indtastningen, står Amigas praktiske editeringsmuligheder til rådighed.

1.2.4 Læsning og editering af programmer

Så snart programmerne har nået en længde på flere skærmsider, vil man sætte pris på de bearbejdningsmuligheder for programmet, som Amiga byder på. Muligheder der kan sammenlignes med et tekstbehandlingsprograms. Det er bedst straks at afprøve enkelthederne på en længere programtekst. Til det formål loades programmet LISTME fra skuffen BASIC-Demos (se Kapitel 1.3.3).

Vi skal finde ud af, hvordan man kommer til et bestemt sted i listningen. Hertil aktiveres List-vinduet, idet man klikker med musen et tilfældigt sted i vinduet. Straks er spørgsmålet besvaret. På det sted, hvor musens pil befandt sig, da der blev trykket på denne valgknap, står nu tekst-cursoren.

Her kan nu indføjes en tekst. Da programmet ikke skal gemmes, er det ikke muligt at gøre noget forkert. Man kan også bruge cursortasterne, til at nå det ønskede sted i programmet. Cursor-tasten med pil til venstre bevæger cursoren til venstre. Prøv det helst med det samme. Hold tasten nede med fingeren. Så snart man er nået til den venstre begrænsning af vinduet, lyder der en advarselstone. Længere går den altså ikke. Tryk nu cursor tasten med pil mod højre. Der lyder ikke nogen advarselslyd, når cursoren kommer til den højre kant af vinduet, men noget andet. Hele teksten forskydes mod venstre. Det sker så længe, som der er mere tekst, men cursoren kan bevæges endnu længere. Først efter en linielængde på 255 tegn lyder advarselslyden igen. Den højre grænse er den maksimale længde af en Basic-linie.

Det er ikke nødvendigt, hver gang at køre hen ad en hel linie med cursoren, for at nå sit mål. Fire tast-kombinationer hjælper med at spare tid, idet de springer over større afstande. Ved kombinationer af to taster, skal man altid først trykke den førstnævnte tast, fastholde den og så trykke på den anden.

> ALT-pil mod højre: sættert cursoren på liniens ende
> ALT-pil mod venstre: sætter cursoren på liniens begyndelse
> SHIFT-pil mod højre: sætter cursoren ved begyndelsen af den sid ste fjerdedel af en programlinie
> SHIFT-pil mod venstre: sætter cursoren ved slutningen af den første fjerdedel af en programlinie

De to cursor-taster "pil opad" og "pil nedad" bringer cursoren en linie ad gangen opad eller nedad. Men det har læseren sikkert allerede gættet. Hvis man kommer til den øverste eller nederste kant af vinduet, med en af disse to taster, er det ikke slut med det. Listningen begynder at scrolle opad eller nedad. Først når man er kommet til programmets begyndelse eller slutning, hører

man igen tonen som tegn på, at man ikke kan komme længere. Med et længere program er metoden for omstændelig. Her hjælper Amiga-Basic med fire tastkombinationer:

Alt-pil opad: sætter cursoren i første programlinie
ALT-pil nedad: sætter cursoren i sidste programlinie
SHIFT-pil opad: bladrer et vindue ad gangen baglæns gennem listningen
SHIFT-pil nedad: bladrer et vindue ad gangen fremad gennem listningen

Nu, hvor vi ved, hvordan man uden problemer kommer til et udvalgt sted i programmet, vil vi begynde på nogle editeringsforsøg. Først gør vi list-vinduet så bredt som muligt. Så går vi med cursortasterne til et vilkårligt sted i programmet, som vi vil ændre. Tegn til venstre for cursoren kan slettes med BACKSPACE-tasten. Men man skal passe på ikke at slette mere, end man ønsker. Det gør ikke så meget i vort øvelsesprogram, men i rigtige programmer kan man lave en del skade på den måde. Hvis man holder BACKSPACE for længe nede, så repeterer tasten hurtigere, end Basic kan slette tegnene. Således kan det ske, at der bliver ved med at slettes tegn, længe efter at man har sluppet tasten.

Tegn til højre for cursoren kan slettes med DEL-tasten. Ellers opfører DEL sig nøjagtig som BACKSPACE. Selvfølgelig kan man ikke alene slette tekst, der hvor cursoren er, men også indtaste ny tekst. Prøv det! Vil man tilføje en hel linie, går man med cursoren til enden af den foregående linie og trykker RETURN. Straks har man en ny, tom linie.

Dernæst skal vi se, hvordan man udvælger bestemte områder af teksten. Sæt musens pil foran et bogstav, tryk på valgtasten, hold den nede og bevæg pilen nogle bogstaver mod højre. Slip så tasten. Bogstaverne, man kørte hen over, er nu omgivet af rødt. sådan kan man udvælge enkelte bogstaver eller tal eller hele linier. Basic kender endda en forkortelse til udvælgelse af et enkelt ord: man kører med musepilen hen over ordet og klikker to gange hurtigt efter hinanden. Man kan også vælge flere linier på en gang. Klik i den første linie af det område, der skal udvælges, og kør så med nedtrykket musetast nedad. I den linie, hvor man vil stoppe udvælgelsen, slipper man så tasten. Herunder behøver man ikke at tage hensyn til vinduets over- og underkanter, men kan køre videre opad eller nedad, til teksten begynder at forskydes.

Et område, der er udvalgt (selekteret) på denne måde, kan man slette på én gang med BACKSPACE, eller bearbejde med Edit-menuens menupunkter.

Med menupunktet Cut slettes det markerede område i List vinduet og gemmes i et midlertidigt lager. Tastkombinationen "Amiga-X" har samme virkning som Cut. (Når der i det følgende henvises til en tastkombination med en Amiga-tast, menes der, med mindre andet udtrykkelig er angivet, altid den højre Amiga-tast.)

Det er bedst at prøve med det samme i listningen af programmet LIST-ME. Et program-område, der er skåret ud på den måde, er ikke gået tabt, men befinder sig i det såkaldte midlertidige lager. Man kan indsætte dets indhold på et andet sted med kommandoen Paste. Hvis der på dette tidspunkt er selekteret et område, erstattes dets indhold af det midlertidige lager. Der er også en forkortelse for Paste: "Amiga-P".

Menupunktet Copy virker ligesom Cut, men det valgte område slettes ikke. Også til den kommando findes en tastaturforkortelse: "Amiga-C".

Som det ses under de hidtil omtalte menupunkter, er det muligt i stedet at kalde dem med en tastkombination. Visse menukommandoer kan dog gives på endnu en måde. For bedre at kunne demonstrere det, skal programmet Picture loades fra skuffen BASICDemos.

At man kan åbne List vinduet eller hente det frem med Show List, er allerede omtalt. Tastkombinationen "Amiga-L" har samme virkning. Man kan også indtaste LIST i direkte funktionsmåde og åbne List-vinduet, selvom der ikke er noget program og dermed heller ingen listning til stede:

LIST linie-linie,filangivelse
det aktuelle Basic-program listes på udlæsningsenheden

Når man kun vil se et bestemt område, skal det angives, hvilke hopmærker det skal ligge imellem. (Hopmærker forklares senere.) Indtast følgende til programeksemplet Picture:

list -CheckMouse

Programmet bliver kun listet til hopmærket CheckMouse. LIST kan også bruges i program-funktionsmåde. Så skal tredje programlinie ændres til:

CLD:LIST

og programmet startes, hvorefter man får listningen, og kan så gå videre med programmet.

Der findes en Basic-kommando til ændring af programmer. Når dele af et program ikke længere behager en, kan man slette dem med følgende kommando, uden at have udvalgt dem med musen:

DELETE mærke1-mærke2
sletter det angivne linieområde

Prøv det først i direkte funktionsmåde:

delete CheckMouse-MovePicture

Området mellem hopmærkerne CheckMouse og MovePicture, inklusive begge hopmærker, er nu slettet.

1.2.5 Styring af programudførelsen

Den omfangsrigste af alle Amiga-Basic menuerne er Run-menuen. Seks menupunkter hjælper med at bestemme, hvorledes programmet skal udføres. Det er det vigtigste værktøj til lokalisering af programfejl.

Med Start igangsættes det netop bearbejdede program. Tastkombinationen "Amiga-R" har samme effekt. Der er dog også en Amiga-Basic kommando, der loader og starter et program:

RUN
RUN filnavn,R
starter et program, resp. loader og starter det

Hvis parameteren R er angivet, forbliver allerede åbnede filer åbne. Uden R lukkes åbnede filer, før programmet startes.

Indtast nu i Output-vinduet:

run "BASICDemos/Picture"

Programmet Picture loades og startes straks.

Programmet er opbygget, så det ikke afsluttes. Man må altså gribe ind udefra, for at standse det. Man kan afbryde et kørende program med menupunktet Stop. Amiga-Basic vender så tilbage til direkte funktionsmåde. Her er der to tastkombinationer, der gør det samme, "Amiga". (punktum) eller CTRL-C. Man kan også indbygge en Basic-kommando af samme navn (og samme funktion) i et program:

STOP
afbryder programudførelsen og vender tilbage til direkte funktionsmåde

Det er en kommando, man mest får brug for i testfasen af et program. Således kan man i direkte funktionsmåde fastslå den aktuelle værdi af en variabel, og om nødvendigt ændre den. Tilføj nu et STOP bagved LINE-sætningen i programmet Picture, og vælg atter kommandoen START. Programmet standses, efter at det hvide rektangel er tegnet.

Fordi programmet kun er blevet afbrudt, kan man få det til at køre videre, som om der ikke var sket noget. Med menupunktet Continue, eller med Basic-kommandoen CONT i direkte funktionsmåde, fortsætter man et afbrudt program.

CONT
fortsætter et program fra afbrydelsesstedet

Er der foretaget en ændring i et program, kan det ikke længere fortsættes. I dette tilfælde, og når der af programtekniske grunde ikke er mulighed for at fortsætte, vises dette af en requester (Can't continue). Så skal programmet startes på ny.

Med Suspend fra Run-menuen eller med tastkombinationerne "Amiga-S" og CTRL-S kan man standse et Basic-program forbigående, uden at det bliver afbrudt. Det vender ikke tilbage til direkte funktionsmåde, og kan derfor heller ikke bearbejdes. Det standser, indtil man trykker på en anden tast end "Amiga-S". Prøv denne nye kommando på programmet Picture. (Husk at fjerne STOP-kommandoen!)

Nu kommer vi til Run-menuens vigtigste kommandoer til fejlsøgning. Det er nemmest at finde fejl, når man med listningen foran sig kan følge, hvordan programmet afvikles. Det er ikke noget problem med Amiga-Basic. Vælg ganske enkelt Trace On fra menuen Run. Hvis List vinduet er synligt, vises nu hver enkelt sætning indrammet af et rødt rektangel, medens Basic udfører den. Trace Off frakobler igen programeftersporings-funktionsmåden. Det rektangel, der indrammede den sidst udførte kommando, forsvinder. Til disse to menu-kommandoer findes også tilsvarende Basic-kommandoer, der kan indsættes i direkte- og program-funktionsmåde:

TRON
TROFF
kobler programeftersporingen ind og ud

Dette par kommandoer giver en mulighed for nøje at følge sine Basic-programmer. Afbryd nu Picture-programmet. Kald først Trace On og så Start. Derefter aktiveres List-vinduet. Nu kan man følge ellipsens fremkomst i Output-vinduet og i List-vinduet følge udførelsen af de enkelte kommandoer. Herigennem bliver forløbet af Basic-programmet også betydeligt langsommere. Man kan let følge kørslen, og gribe ind over for eventuelle fejl.

Amiga-Basic giver en endnu finere kontrol med programmet, så at sige et Super-Trace. Kommandoen Step udfører nøjagtigt en linie af programmet og standser så. Også her bliver den pågældende sætning indrammet med rødt, når List-vinduet er åbent. Tastatur-kombinationen "Amiga-T" har samme virkning som Step i Run-menuen.

Man bør afprøve dette menupunkt på programmet Picture. Afbryd programmet og indtast END i direkte funktionsmåde. Kald så Step frem fra Run-menuen, og åbn List vinduet. Hver gang man på ny kalder Step, bliver en ny kommando i listningen udført og markeret med et rødt rektangel. Tastkombinationen "Amiga-T" er naturligvis hurtigere end at vælge et menupunkt.

1.3 Loade og save (gemme) programmer

Basic-menuen Project indeholder de fleste af de menupunkter, man behøver til en ordentlig administration af programmer på disketten. (Hertil findes selvfølgelig også Basic-kommandoer.) For ikke ved en forglemmelse at miste sit bedste Basic-program, når man går ud af Basic, spørger Amiga-Basic med en requester, om man ikke vil gemme programmet. Man svarer ved at klikke det ønskede svar i et af de tre felter.

1.3.1 En rute fører gennem filerne

Før man begynder at loade og save programmer, må man først kende til behandlingen af filer og directoryer ("filkataloger"). Læg Amiga-Basics vinduer om bagved og kig på Extras-diskettens vinduer. Her finder man skuffen BASICDemos, hvorfra vi allerede har loadet nogle programmer. Denne skuffe, og alle de øvrige skuffer, svarer til en directory. Befinder der sig endnu en skuffe i denne skuffe, ville directoryen være en underdirectory under førstnævnte directory. Dette eksempel kan man videreføre, så længe man har lyst. I skuffen findes programmer og filer (f.eks. ball). Alt det er filer inden for skuffens directory. Den øverste directory er altid diskette-directoryen, der også betegnes som rod- (root) directory.

En komplet beskrivelse af det sted, hvor en bestemt fil findes, betegnes som filens rutenavn og er sammensat således:

Diskettenavn eller drev: en eller flere directory/navne

Lad os nu vende tilbage til Basic. Vil man vide, hvad der er på disketten, skriver man følgende kommando i direkte funktionsmåde:

Filens filnavn
Indholdet af en directory vises

Hvis man ikke angiver nogen fil, får man listet alle filer i den aktuelle directory. Ønsker man at se filerne i en anden directory, skal man angive et rutenavn, som lige omtalt. Med

files "df0:BASICDemos"

får man f.eks. indholdet af skuffen BASICDemos. Med

files "Workbench:Demos"

kan man se indholdet af skuffen Demos på Workbench-disketten. Når man vil arbejde meget med en bestemt directory, der ikke er den aktuelle, kan det blive ret omstændeligt at skulle indtaste hele rutenavnet. En Basic-kommando overtager dette arbejde, idet den gør en vilkårligt ønsket directory til den aktuelle:

CHDIR rute
gør rute til den aktuelle directory

Vil man gøre skuffen BASICDemos til den aktuelle directory, indtastes i direkte funktionsmåde:

chdir "df0:BASICDemos"

1.3.2 New

Med New fra Project-menuen slettes programmet i hovedhukommelsen samt alle variabler. Programlistningen i List vinduet slukkes ligeledes. Efter udførelse af New sletter Basic-fortolkeren tillige Output-vinduet og vender tilbage til direkte funktionsmåde. Hvis programeftersporingen skulle være aktiveret, frakobles den også. New vil man altid anvende, når man skal begynde på et nyt program. I stedet for menu-optionen New kan man indtaste en Basic-kommando i direkte funktionsmåde:

NEW
Programmet og alle dets variabler slettes fra hukommelsen

1.3.3 Loadning af et program

Man kan loade et program på forskellige måder. Her skal vi lære tre at kende.Det letteste er at loade et program, ved at starte dets program-ikon med et dobbeltklik på Basic-disketten. Metoden er ganske vist den langsomste, hvis man allerede er i Basic, fordi man så skal gå ud af Amiga-Basic.

Den anden er Project-menuen. Med kommandoen OPEN kan man loade et program fra disketten til hukommelsen. Først slettes alle åbne filer, samt det netop bearbejdede program. Prøv engang. Der åbnes et kommunikationsvindue (requester), som opfordrer til loadning af et program:

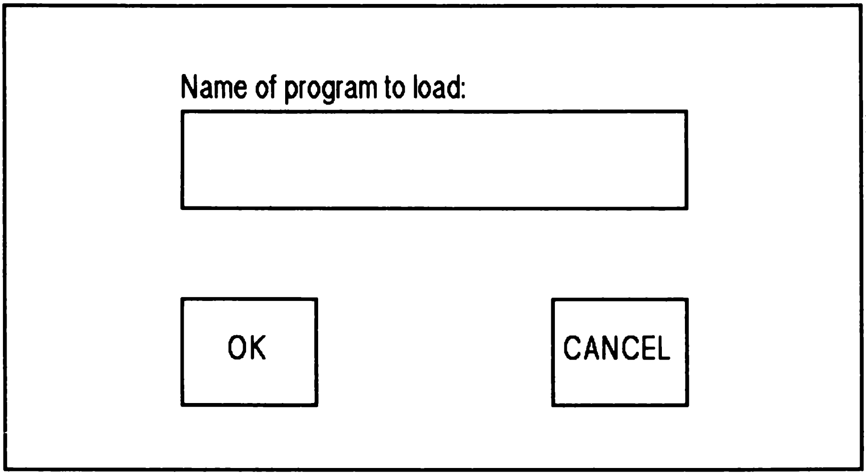


Fig 1.1
Vi vil loade programmet Picture fra skuffen BASICDemos. Klik i feltet for programmets navn og indtast:

BASICDemos/Picture

Tryk så RETURN eller klik med musen i OK-feltet. Programmet loades nu. Hvis List-vinduet er åbnet, vises listningen straks.

Den tredje måde at loade et program på, byder igen på en Basic-kommando:

LOAD filangivelse,R
loader og starter et program fra diskette eller harddisk

Når parameteren R indtastes, loades og startes programmet. Ellers vender fortolkeren tilbage til direkte funktionsmåde efter loadningen.

Eksempel load "BASICDemos/Picture"
eller: load "df0:BASICDemos/Picture",R

1.3.4 Save

Med Save fra Project-menuen gemmes det aktuelle program under samme navn, som man før har loadet det med. Herunder bliver programmet gemt i det format (se næste kapitel), hvori det tidligere blev loadet. Hvis man har oprettet et nyt program, eller vil gemme et program under et andet navn, kan man ikke anvende denne option. Man må kalde kommandoen Save As.

Kommandoen har også andre anvendelsesmuligheder. Det oprindelige program, der befinder sig på disketten, bliver overskrevet med et nyt program, som man lagrer under samme navn. Hvis man vil bevare det gamle program, når man gemmer en ny version af det, vælger man optionen Save As. Endnu et tip i den forbindelse: åbn altid List vinduet før saving. Derved kan man se, om programmet endnu befinder sig i hukommelsen. Hvis man, ved en fejl i programmet, som Basic ikke har opdaget, skulle have slettet sit program, ses det af den manglende listning. Hvis man saver et sådant program, gemmer man "ingenting", og taber sit gamle program.

1.3.5 Save as

Memupunktet Save As fra Project-menuen fungerer ligesom Open, med den ene forskel, at det navn, man her indtaster, denne gang ikke tjener til at loade et program, men til at save det under samme navn. Når man kalder Save As, fremkommer følgende kommunikationsvindue:

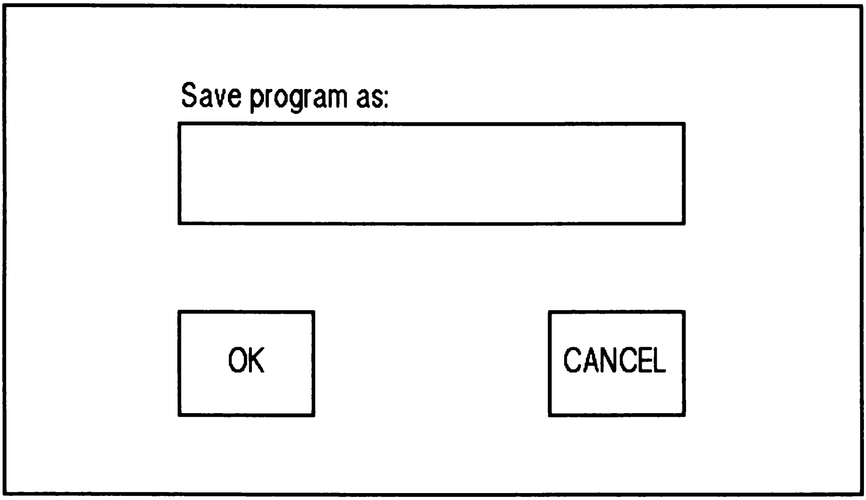


Fig. 1.2

tekstfeltet foreslås altid navnet på det sidst loadede program, i vort tilfælde BASICDemos/Picture. Hvis programmet er nyoprettet, eller der i mellemtiden er blevet valgt New, forbliver tekstfeltet tomt. Klik nu med musen i feltet og erstat ordet Picture med Picture3. Når man trykker RETURN eller klikker i OK-feltet, udføres indtastningen. Programmet gemmes. Hvis man vil være sikker, skal man lægge Basic-vinduet bagest, og undersøge indholdet af skuffen BASICDemos.

Med Save As gemmes programmer i et komprimeret format. Vil man gemme et program i et andet format, skal man indtaste Basic-kommandoen SAVE i direkte funktionsmåde:

> **SAVE filnavn,A,P,B**
> gemmer et Basic-program i valgfrit format på diskette

De enkelte options betyder

- B Programmet gemmes binært. Det er standard og kan derfor udelades.
- P Programmet bliver gemt med beskyttelse. Man skal være forsigtig med denne option. Et program, der én gang er gemt således, kan ikke ændres.
- A Programmet gemmes som ASCII-tegn (tekst).

Læseren undrer sig måske over, hvornår man vælger at gemme et program enten binært, eller som ASCII-tegnstrenge. Grunden ligger i hukommelsesforbruget, der er lavest for et binært lagret program. Til MERGE-kommandoen, der gennemgås senere, behøver man option A. Og P er praktisk, hvis man vil beskytte programmet mod at blive ændret.

1.3.6 Quit

Med kommandoen QUIT kan man afslutte arbejdet med Amiga-Basic. QUIT lukker alle åbne filer og Basic-vinduer, og man lander i Workbench eller i CLI-vinduet, hvis det var der, man startede. I stedet for QUIT kan man benytte Basic-kommandoen SYSTEM:

> **SYSTEM**
> Programmet vender tilbage til Workbench (eller CLI)

1.3.7 Ombenævnelse og sletning af programmer

Vil man give et program et andet navn, kan man benytte menupunktet Rename fra Workbench eller en Basic-kommando:

> **NAME filnavn1 AS filnavn2**
> filen filnavn1 får det nye navn filnavn2

I tidens løb får man samlet nogle programmer, man ikke mere vil gemme på. Dem kan man også slette fra disketten med en kommando:

> **KILL filnavn**
> sletter den angivne fil på diskette eller harddisk

1.4 Styring af programforløbet

Kvaliteten af en Basic-dialekt viser sig i antallet af styringssætninger til fastlæggelse af programforløbet. Amiga-Basic har mange. Men antallet er i sig selv ikke nok til en effektiv programmering. Det er også vigtigt, hvordan programdele kan sammenfattes til blokke, og hvordan en programlinie er opbygget, resp. kan forarbejdes af Basic. Man kan indtaste linienumre i Amiga-Basic, eller lade være. Basic-fortolkeren udfører linierne i den rækkefølge, de er indtastet i (med mindre styringssætninger kræver noget andet) og ikke i den rækkefølge, nummereringen angiver. Linierne sorteres heller ikke i nummerorden.

Indtaster man først linie 500 og så linie 10, bliver de stående i samme rækkefølge i listningen. Linienumrene er kun en mulig hopadresse til forgreningsoperationer; men det er bedre at anvende et særligt hopmærke til slige formål.

1.4.1 Labels

Hopadresser i et program kan forsynes med læselige tekster, der kaldes labels. Sådanne programmærker eller hopmærker kan også benyttes til afmærkning af vigtige programafsnit (strukturering). Et mærke skal altid stå i begyndelsen af en linie. Dets længde er begrænset til 40 tegn. Dèt første skal være et bogstav og det sidste et kolon. Kolon'et i slutningen af en label skal være der, fordi det kendetegner et hopmærke. Lignende tekster uden et kolon opfattes af Basic som variabler. Af lignende grunde må der ikke forekomme mellemrum i et hopmærkes navn eller mellem navnet og kolon'et. Man kan udforme hopmærkerne helt vilkårligt, men helst formålstjenligt. En formålstjenlig betegnelse viser, hvad progammet skal gøre, og sparer forklaringer.

1.4.2. GOTO, GOSUB

Man kan hoppe til et programmærke med GOTO-sætningen:

GOTO mærke
ubetinget programforgrening til et hopmærke/linie

Overalt hvor Basic-fortolkeren møder en GOTO-kommando, søges linien, der begynder med pågældende mærke, og fortsætter programmet der. Med det følgende eksempel skriver man uafbrudt en tekst i øvre venstre hjørne af skærmen:

```
start:
CLS
PRINT "Amiga Computer"
GOTO start
```

Denne mest kendte kommando til styring af programforløbet, bør benyttes mindst muligt. Den fører ofte til, at man hopper på kryds og tværs i programmet, så at man let taber oversigten. Vil man senere ændre i programmet, sker det, at man ikke kan finde rundt i det. I det følgende kommer langt bedre kommandoer til opbygning af en ordentlig, overskuelig programstruktur.

I ovenstående eksempel lærte vi en ny kommando at kende:

CLS
sletter indholdet af det aktuelle vindue, anbringer cursoren i HOME-position (øverst til venstre)

Den anden kommando til styring af programforløbet er sikkert også allerede kendt:

GOSUB hopmærke
hopper til en subrutine, der senere vender tilbage til den linie, hvor GO-SUB'et stod
RETURN hopmærke
vender tilbage fra en subrutine til det sted, hvorfra den blev kaldt (med GOSUB)

Med GOSUB kan man hoppe til en subrutine (GOSUB = GO to SUBroutine), hvorfra man kan kalde en subrutine osv. Det er vigtigt, at hver subrutine eller videreforgrening afsluttes med et RETURN. Hvis man altså vender tilbage til udganspunktet uden et RETURN, bringer man forvirring i hele programforløbet. Derfor skal man kun med forsigtighed anvende GOSUB'er, der forlader en subrutine.

En subrutine indsættes, når et programstykke hyppigt skal genanvendes. Det forkorter programmet, sparer unødvendigt skriveri og er desuden mere overskueligt.

Subrutiner skal helst anbringes i slutningen af programmet, eller samlet forrest i programmet, og der skal sørges for, at de ikke kan gennemløbes utilsigtet (uden GOSUB).

Det følgende program viser, hvordan man kan kalde en tidsløkke, der forsinker programforløbet, med GOSUB. FOR-NEXT løkken til forgreningen beskrives i næste afsnit. (I følgende afsnit af bogen findes mange eksempler på subrutiner, der vil vise sig mere praktisk anvendelige.)

REM langsamer
'P1.4.2-1

I programmet findes en styringssætning, der forhindrer at programmet kaldes bag PRINT og således tilgås ukontrolleret (uden GOSUB):

END
afslutter programmet, lukker alle åbnede filer

Står sidste kommando i den sidste linie af et program, behøver man ikke at indtaste END; men hvis der følger subrutiner bag programmets "egentlige" slutning, så er END absolut nødvendig.

De første to linier i programeksemplet har ingen virkning, men overses. Det er kommentarer (engelsk REMarks):

REM kommentartekst
bemærkninger efter et REM overses

I stedet for REM kan man anvende apostroffen ' (uden shift, til venstre for RETURN-tasten). Efter et REM bliver der ikke udført kommandoer, der står på samme linie.

En interessant variant af GOTO- og GOSUB-sætningerne tillader at hoppe programmeret til forskellige mål:

ON tal GOSUB hopmærke,hopmærke,...
ON tal GOTO hopmærke,hopmærke,...
forgrener afhængigt af tal til det pågældende hopmærke.

Som tal anvendes værdier mellem 1 og 255. Hvis tallet f.eks. er tre, forgrenes til det tredje hopmærke. Dette kommandopar er ideelt til programmering af menuforgreninger. Man vil i denne bog se det anvendt i næsten alle programmer, der bruger menuer.

Følgende eksempel viser kommandoens funktion:

REM Textverzweigung
'P1.4.2-2

Ved første gennemløb får x værdien 1, og der forgrenes til den sætning, som står på første position efter ON x GOTO. Programmet skriver den første tekst og vender med GOTO tilbage til starten. Der får x værdien 2 osv.

1.4.3 FOR-NEXT løkker

FOR variabel=x TO y STEP z
...
NEXT variabel, variabel,...

løkke med et fastlagt antal gennemløb til flerdobbelt bearbejdning af kommandoer og funktioner

Med løkken kan man gøre mangt og meget. Variablen får tildelt startværdien x under det første gennemløb. Trinvidden af gennemløbene fastlægges med STEP. Hvis man f.eks. indtaster

```
FOR t = 1 TO 10 STEP 2:PRINT t:NEXT t
```

gennemløbes løkken kun fem gange. Variablen t får herunder værdierne 1, 3, 5, 7, 9 efter hinanden. Så forlades løkken, fordi værdien 10 (dvs. grænseværdien y) ville blive overskredet ved næste gennemløb. Hvis trinbredden z ikke angives, antages den at være 1. Man kan også gennemløbe løkkeværdierne med faldende værdier. Så skal startværdien naturligvis være større end grænseværdien. Det er endda tilladt med decimalbrøker. Den følgende løkke gennemløbes således seks gange. Variablen j får værdierne 9, 7.5, 6, 4.5, 3 og 1.5.

```
FOR j = 9 TO 1.5 STEP -1.5:PRINT j:NEXT
```

Man kan anbringe flere FOR-NEXT løkker inden i hinanden (parentetisk). Så skal variablerne blot have forskellige navne.

I det følgende eksempel anbringer vi to løkker inden i hinanden:

REM Plus
'P1.4.3-1

For hvert gennemløb af den ydre løkke gennemløbes den indre løkke 20 gange, i alt altså 20 * 100 gange. På skærmen ses additionen af variablerne j i den ydre og variablerne i i den indre løkke. Den tredje løkke tjener til forsinkelse, så resultatet af PRINT-sætningen kan nå at blive aflæst.

Det sidste eksempel på FOR-NEXT løkken tegner en trekant:

REM trekant
'P1.4.3-2

Løkkkevariablen y indeholder værdierne til linierne. Variablen i i den indre løkke er ansvarlig for antallet af tegn på de enkelte linier, men bruger hertil også y fra den ydre løkke. Anbringelsen af cursoren sker med følgende sætning:

LOCATE linie,spalte
anbringer tekstcursoren i det aktuelle vindue

Værdierne af linie og spalte skal være heltal, der er større end nul. Måske har læseren i vore første små programmer bemærket, at nogle linier er indrykket. Amiga-Basic tillader den form for formattering ved løkker. Man kan straks se på en listning, hvilke kommandoer, der hører til inden i en løkke, også når der er flere løkker inden i hinanden. Disse indrykninger, der kan fås med TAB-tasten, gør programmerne meget mere overskuelige.
med småt, blev de automatisk forvandlet til store bogstaver, når man skrev videre på den næste linie. Derfor er det let at genkende alle Basic-kommandoer i programmet. Det er ligegyldigt, om man skriver dem med småt eller stort eller en blanding heraf.

Dette tilsyneladende pjat er en stor hjælp til rigtig indtastning af Basic-kommandoer. Man bør altid se efter, hvad man har indtastet. Hvis Basic-ordene ikke fremkommer med store bogstaver, efter tryk på RETURN-tasten, så er der begået en anslagsfejl. I så tilfælde skal man afprøve, om man har skrevet alle kommandoer i den pågældende linie rigtigt, og om der er et mellemrum efter hvert af dem.

Der skal være mellemrum efter næsten alle Basic-kommandoer.

For ikke at have besvær med store og små bogstaver under programmeringen, har Amiga-Basic's konstruktører indbygget en yderligere fordel. Prøv i programmet at skrive linien "GOTO start" med et stort "S", så fremkommer ordet Start også med stort ved hopmærket. Denne praktiske effekt kan man også se på variabler og feltvariabler. Den samme variabel skrives altså altid på samme måde i hele programmet. Det samme gælder for indtastninger i direkte funktionsmåde i output-vinduet.

1.4.4 IF-sætningen

Nu kommer vi til den vigtigste sætning i Basic. Med den kan man vurdere de forskellige betingelser i et program, eller en indtastning, og efter resultatet træffe forskellige afgørelser. Lad os se på kommandoerne:

> **IF udtryk GOTO label ELSE sætning**
> Hvis udtryk er sandt, sker et spring til label'en GOTO. Hvis ikke, udføres sætningen bagved ELSE
> **IF udtryk THEN sætning1 ELSE sætning2**
> Hvis udtryk er sandt, udføres sætning1 bagved THEN, er det falsk, udføres sætning2 bagved ELSE

IF-THEN sætningen må i denne form højst fylde en linie (255 tegn). Det kan ofte virke som for lidt. Derfor kender Amiga-Basic endnu en form for IF-

THEN-Else , sætning,hvor der kan stå vilkårligt mange sætninger mellem THEN og ELSE.

```
IF udtryk1 THEN
Linier med sætninger
ELSEIF udtryk2 THEN
Linier med sætninger
ELSEIF udtryk3 THEN
...
ELSE
Linier med sætninger
END IF
```

Hermed afprøver Basic flere betingelser (udtryk). Hvis et af dem er sandt, bliver kommandoerne i de følgende linier udført, indtil der kommer et ELSEIF eller END IF. Så springer Basic hen bag END IF-sætningen. Hvis ingen af udtrykkene er sande, udføres derimod kommandoerne mellem ELSE og END IF. For alle de forskellige versioner af IF-sætningen gælder et udtryk som sandt, når dets værdi ikke er nul, og som usandt eller falsk, når dets værdi er nul (se Kapitel 1.6.2).

Før vi tydeliggør de enkelte sætninger med et par eksempler, gøres opmærksom på, at ELSEIF skal skrives i ét ord! Der står ganske vist noget andet i Basic-manualen - her er et mellemrum mellem ELSE og IF - det er forkert!

Lad os se på den første udgave af IF-sætningen:

```
c=1 : IF c GOTO udlæs ELSE PRINT "falsk"
END
udlæs: PRINT "Værdien er korrekt"
```

Vi anmoder altså om:

Hvis c er sand, så gå til udlæs, ellers skriv "falsk"

Da vi har opgivet c som 1 (sand), hopper programmet til udlæs. Prøv det engang med c=0. Det andet eksempel anvender den anden form for IF-sætning:

```
a=1 : b=0
PRINT "a=1, b=0"
IF a THEN PRINT "a=sand" ELSE PRINT "a=falsk"
IF b THEN PRINT "b=sand" ELSE PRINT "B=falsk"
```

Disse to simple eksempler er lette at forstå. I det næste eksempel vises en mere kompliceret IF-blok. Vi vil fylde skærmen med tegn på nær et hul i det øverste venstre hjørne. Herunder skal vi vurdere nogle betingelser i programmet (sammenligningsoperatorerne finder man forklaret i afsnit 1.5):

REM freilassen
'P1.4.4-1

Hvis man læser forklaringerne i de enkelte linier opmærksomt, vil man hurtigt gennemskue, hvordan sådan en blok af If-sætninger kan opbygges. Nu er der ingen hindring for egne forsøg, måske med endnu flere sætninger inden i hinanden.

1.4.5 WHILE-WEND løkken

Den giver interessante muligheder i programudformningen:

```
WHILE udtryk
Linie med sætninger
WEND
```

...
et vilkårligt antal sætninger udføres i en løkke, sålænge udtryks betingelser er sande.

Når Basic møder en WHILE-sætning, afprøves det, om udtryk er sandt. Hvis ja, udføres alle sætninger indtil WEND. Hvis ikke, springes de pågældende sætninger over, og programmet fortsætter bagved WEND. Hver gang Basic når til WEND, hopper det tilbage til den tilhørende WHILE-sætning og afprøver den pȧny. Er udtryk igen sandt, udføres sætningerne igen osv. Inden for en WHILE-WEND sætning ændres variablerne i reglen således, at betingelsen for WHILE-sætningen (udtryk) før eller senere bliver falskt - ellers fortsættes i en uendelig løkke.

WHILE-WEND sætninger kan - som IF-sætninger - også anbringes inden i hinanden; men kig først på dette enklere eksempel:

```
REM alfabet
a=65
WHILE a[ PRINT CHR$(a);
a=a+1
WEND
```

Programmet skriver det store alfabet på skærmen. Variablen a får først værdien 65. Under hvert gennemløb af løkken forhøjes værdien med 1. Så snart

a har nået værdien 91, forlades løkken. Tallene i variablerne er kodetal for den internationale tegnkode ASCII (udtales "aski"). CHR$ giver det bogstav, der svarer til et sådant tal.
Nu kommer et eksempel, hvor WHILE-WEND sætningen dukker op i parentetisk form:

REM Pfeil
'P1.4.5-1

Programmet får en lille pil til at vandre fra venstre til højre side af skærmen. Der vender pilen med et "pip" og går tilbage til venstre side en linie længere nede. Det hele gentages, indtil den nederste kant af skærmen er nået.

Den ydre WHILE-WEND løkke tæller variablen y én højere for hveranden linie. Når y har nået værdien 21, forlades denne løkke, og programmet begynder forfra. Den første indre WHILE-WEND-løkke anbringer cursoren i position, og skriver en pil ().Derefter sørger en lille tidsløkke for, at pilen kan ses bedre og længere. Den følgende ASCII-kode 8 svarer til BACKSPACE-tasten. Dermed slettes pilen. ASCII-koden 32 skriver et tomt mellemrum, og så er vi igen på den gamle position. Variablen x forhøjes med én. Hvis denne variabel har værdien 79 for den højre billedkant, så forlades WHILE-WEND-løkken. Der udløses et pip og tælles en linie videre. Den anden indre WHILE-WEND-løkke bringer en venstre pil til venstre billedkant. Så mangler blot forklaringen på det akustiske signal:

> **BEEP**
> frembringer en advarselstone og et kort blink af billedskærmen

Som det ses, er brugen af WHILE-WEND løkken ret simpel. Hvad den videre kan bruges til, viser næste kapitel.

1.4.6 Hvem vækker Tornerose?

Her skal vi lære et programforløb at kende, som kan styre f.eks. et helt 10-siders Basic-program:

> **WHILE 1:SLEEP :WEND**

Denne linie, på kun få bogstaver, kan styre forløbet af et vilkårligt stort program! Det skal blot på forhånd være forberedt på visse afbydelser. Lad os se nærmere på sove-kommandoen:

SLEEP

Programmet venter, til der sker en af følgende dele

- Der trykkes på musens valgtast
- Der trykkes på tastatur-tasten
- En kollision af grafiske objekter
- Valg af en menutitel
- Udløb af en fastsat tid

Denne liste over begivenheder, der kan forstyrre løkkens søvn, er med en enkelt undtagelse (tastatur-tasten) såkaldte interrupt- (afbrydelses-) begivenheder. Ordet repræsenterer en mængde muligheder for styring af et programforløb. Med denne type begivenheder kan man styre et program ved hjælp af pull-down (nedtrækkelige) menuer, undersøge musetastens tilstand og meget mere

Følgende begivenheder kan programmeres:

- TIMER reagerer på udløbet af en fastsat tidsperiode
- MOUSE reagerer på musens valgtast
- MENU reagerer på musens menutast
- BREAK reagerer på højre Amiga-tast-punktumtast
- COLLISION reagerer på kollision af grafiske objekter

Følgende sætninger overtager styringen:

- BegivenhedON kobler reaktionsevnen ind
- BegivenhedOFF kobler reaktionsevnen fra
- BegivenhedSTOP afbryder reaktionsevnen midlertidigt

I stedet for givenhed skrives et af nøgleordene fra listen ovenfor. Først skal man dog fastlægge, hvorhen programmet skal forgrenes ved en begivenhed:

ON Begivenhed GOSUB label

Det bliver fastlagt i subrutinen, hvad der skal ske, når begivenheden indtræffer. Derefter aktiverer man interrupt-reaktionsevnen med "Begivenhed ON". Tilsammen kan begge sætninger se således ud:

ON MENU GOSUB hopmærke: MENU ON

Så snart "Begivenhed ON" er udført, kontrollerer Basic-fortolkeren før hver ny sætning, om begivenheden er indtrådt. Hvis ja, så forgrenes programmet (uden GOTO og GOSUB) til det hopmærke, der er angivet bag "givenhed ON". Ved hopmærket skal der begynde en subrutine, som ender med RE-

TURN. Dermed hoppes tilbage, hvor begivenheden blev fastslået af Basic, og programmet fortsættes.

Kapitlet giver et indblik i interrupt-begivenhedernes teknik. Flere enkeltheder og eksempler findes i forklaringerne til de enkelte sætninger. Hvordan man kan anvende begivenhederne, vil herigennem blive meget klarere.

1.5 Variabler

I næsten alle de programmer, vi har lært at kende, har der været bogstaver eller ord, som stod i forbindelse med tal. Yderligere mødte vi ofte i forklaringerne ordet variabel, der refererer til disse navne. Det skal vi se på i dette afsnit.

1.5.1 Barnet skal have et navn

Vi har lært, at en tekst, som skal udlæses af PRINT-sætningen, skal omgives af anførelsestegn. Prøv, hvad der sker, når man udelader dem:

print Commodore

I stedet for ordet Commodore får man:

```
0
OK
```

Overrasket? Prøv så at indtaste:

```
Commodore=1+5
print Commodore
```

Denne gang får vi et sekstal, selv om det var ordet Commodore, vi ville se.

Næsten alle har prøvet at håndtere matematiske formler.Heri indsætter man bogstaver i stedet for værdier, der kan variere. Det er netop det, det drejer sig om her, nemlig VARIABLER. I det lille eksempel opfattede Basic ordet Commodore som en variabel, og brugte det til at regne med. Variabelnavne kan nemlig ikke blot være på et eller to bogsatver i Amigaen - som i en del andre computere. Amiga-Basic kan skelne mellem variabelnavne, der er 40 tegn lange. Blot må man ikke anvende en Basic-kommando som variabelnavn, selv om det er tilladt som del af et navn. Test det ved at indtaste:

```
cls = 3
```

Blev De forskrækket af Amiga-Basics fejlmelding, så skru ned for lydstyrken. Det er ikke kun begyndere, som får denne tone at høre. Vi har fået fejlmeldingen Syntax Error (syntaks-, dvs. grammatisk fejl). Slet fejlmeldingen ved at klikke i OK-feltet. Prøv så:

```
clsforetag = 3
```

Denne gang accepteres variabelnavnet med OK.

1.5.2 Variabeltyper

Amiga-Basic skelner mellem

- Tegnkæde- eller strengvariabler og
- numeriske variabler.

Strengvariabler er til alle tekster. De kan maksimalt være på 32767 enkelttegn og kendetegnes af et $ som sidste tegn i navnet.

```
tekst1$ = "Vent venligst"
a$ = "Resultatet er: "
```

Før man tildeler en tegnstrengs-variabel en tekst, indeholder den en tom streng med længden nul:

```
a$ = ""
```

Numeriske variabler har værdien nul, før de tildeles en værdi. Den værdi en variabel har, kan man ændre enten i direkte funktionsmåde, altså med en indtastning, eller med en Basic-kommando, idet man tildeler variablen en ny værdi. Følgende kommando tildeler en variabel en værdi:

LET variabel = udtryk
tildeler en variabel værdien udtryk

Kommandoordet LET kan udelades, som de første eksempler viser. Lighedstegnet = er tiltrækkeligt til at opnå en tildeling. (LET er kun med af historiske grunde, tidligere Basic-fortolkere på andre computere krævede denne kommando.) Man vil derfor næppe finde den i et program.

De numeriske variabler inddeles efter, hvor præcis værdien skal være. Den følgende oversigt sammenfatter alle vigtige punkter vedrørende variabeltyper. Det kan også tjene som et lille opslagsværk, hvis der senere optræder tvivl.

VARIABELTYPER:

Type	Betegnelse	Kendetegn	Hukommelseskrav	Gælder for
INT	kort heltal	%	2 bytes	tal uden decimaltegn fra -32768 til +32767
LNG	langt heltal	&	4 bytes	tal uden decimaltegn fra -2147483648 til +...647
SNG	enkelt præc	!	4bytes	indtil 7 cifre, fra 1.18*10^-38 til 3.4*10^38
DBL	dobbelt præ	#	8 bytes	indtil 16 cifre, fra 2.23*10^-308 til 1.79*10^308
STR	tegnstreng	$	5 bytes	indtil 32767 tegn + længde indrammet af ""

Tabel 1: Variabeltyper og deres kendetegn

Vil man karakterisere en variabel som kort heltal, skal man altså f.eks. skrive:

```
linie%=20
```

Når en variabel ikke har et særligt kendetegn, antager Amiga-Basic, at det er et enkelt-præcisions tal.

Hvis man kun vil anvende én type, eller hovedsagelig én type variabler i sit program, kan man bestemme (definere) for Basic, at alle variabelnavne uden noget kendetegn, ikke skal behandles som enkelt-præcisions tal, men som variabler af den ønskede type.

DEFType Bogstav-bogstav
Fastlægger for alle variabler uden type-endelse og med et bestemt begyndelsesbogstav, hvilken type de skal have

Vil man kun arbejde med variabler af korte heltals-typen, definerer sætningen DEFINT A-Z alle variabler som korte heltal. (Bogstaverne kan enten skrives med stort eller med lille.) Hvis man kun har angivet et bogstav, f.eks. A, så bliver alle variabler, der begynder med A, defineret som korte heltals. Vil man undlade denne sætning, er man nødt til at skrive et kendetegn efter hver variabel. Husk, at variabler af forskellige typer ikke fylder lige meget i hukommelsen. Variabler med unødvendig stor præcision bruger derfor unødvendigt megen hukommelse. Får man alligevel brug for flere typer variabler, så er der ikke noget i vejen for det, trods DEFINT-sætningen. Den gælder jo kun for variabler uden type-kendetegn til slut. Har man med DEFINT defineret alle tal som korte heltal, og alligevel får brug for et tal med enkelt præcision, skriver man et udråbstegn (!) bagved navnet på den pågældende variabel.

Efter al den teori nu et lille eksempel:

REM Variablen
'P1.5.2-1

let af pladser, som kommaet, i tallet foran E, skal forskydes mod højre, for at få udregningens virkelige resultat. Støder man på et tal med minus bag E'et, så er tallet for lille til en nøjagtig repræsentation. Lad os betragte et eksempel på det:

6.191736E+10.

Måske har læseren oplevet noget lignende med en lommeregner. Når et tal bliver længere, end det er forudset for variabeltypen, anvender Amigaen denne type eksponential-repræsentation. Tegnet E skal opfattes således:

6.191736E+10 = 6.191736*10^10 = 61917360000

Tallet bag E er lig med antallet af pladser, som kommaet, i tallet foran E, skal forskydes mod højre, for at få udregningens virkelige resultat. Støder man på et tal med minus bag E'et, så er tallet for lille til en nøjagtig repræsentation. Lad os betragte et eksempel på det:

4.123654E-8 = 4.123654*10^-8 = 0.00000004123654

Tallet bag E skal opfattes som det antal pladser, kommaet i tallet foran E, skal forskydes til venstre.

De forskellige datatyper giver en høj grad af fleksibilitet i programmeringen. Man kan altid vælge den type til en variabel, der egner sig bedst - feks. kræver mindst hukommelse. Det kan forekomme, ât numeriske variabler, der er

gemt med én type præcision, senere ønskes med en anden type præcision.Forvandling af en værdi, fra et præcisionstrin til et andet, kan opnås med de såkaldte "konverteringskommandoer".

v=CSNG(udtryk)
forvandler værdien af udtryk til et enkelt præcisions-tal

Indtaster man f.eks.

a# = 13.12394913:PRINT CSNG(a#)
får man
13.12395

v=CDBL(udtryk)
forvandler værdien af udtryk til et dobbelt præcisions-tal.

Hvis man indtaster

a=25.1:PRINT CDBL(a)
får man
25.10000038146973

Tallet kan naturligvis ikke blive mere præcist efter forvandlingen, end det var før. De ekstra decimaler, der er vist, er uden betydning på forvandlingstidspunktet.

Der findes også to funktioner til at gå den anden vej:

v=CINT(udtryk)
forvandler værdien af udtryk (eventuelt ved afrunding) til et kort heltal

Indtaster man

PRINT CINT(34.5) CINT(35.5) CINT(12345.123456789#)
får man
34 36 12345

v=CLNG(udtryk)
forvandler værdien af udtryk (eventuelt ved afrunding) til et langt heltal

Når man indtaster

PRINT CLNG(34,5) CLNG(35.5) CLNG(123456789.123456789#)

får man 34 36 123456789

De to funktioner CINT og LNG runder op, når heltallet er ulige. Når det er lige, rundes der ned.

1.5.3 En specialitet - feltvariabler

indtil man ændrer dem eller sletter feltet. Heri er der ingen forskel på feltvariabler og andre variabler. For at Amigaen kan beregne hukommelseskravet af felter, skal de "dimensioneres" før anvendelsen.

b(0)	b(1)	b(2)	b(3)	b(4)	b(5)	b(6)	b(7)	b(8)

Tabel 1.1 Opbygningen af en feltvariabel

Tildeles feltvariablen værdier (f.eks. b(0)=8 og b(6)=43, gemmes disse, indtil man ændrer dem eller sletter feltet. Heri er der ingen forskel på feltvariabler og andre variabler. For at Amigaen kan beregne hukommelseskravet af felter, skal de "dimensioneres" før anvendelsen.

DIM SHARED variabel(elementer),variabel(elementer)
fastlægger det maksimale antal elementer, som en feltvariabel må indeholde, og reserverer den nødvendige mængde hukommelse.

Før man anvender feltvariabler, skal man altid benytte DIM-sætningen. Computeren tillader maksimalt 11 elemeneter uden dimensionering. Hvis man bruger tilføjelsen SHARED bag DIM, dimensioneres variablerne som "globale felter". De opbevares også, når det program, som sætningen står i, kalder et andet (under-) program. Uden SHARED ville alle variabler blive slettet.

Før vi kigger på et lille program til illustrering heraf, må vi vide, hvordan man får værdier ind i et programs variabler. Der er hovedsagelig to metoder til det: værdierne udregnes af programmet, eller programmets bruger spørges om værdierne, og han indtaster dem. Da en tredje metode er anvendt i det følgende program, bør man først kigge på den sætning, der er beregnet på det:

INPUT "tekst"; variabel1, variabel2,...
INPUT ; variabel1, variabel2
udlæser tekst og læser så variabelværdierne fra tastaturet.

Teksten i INPUT-sætningen er et såkaldt "klartegn". Den siger til brugeren, hvad der skal indlæses. Så følger en liste af variabler, som optager de indtastede værdier. Hvis man i stedet for et semikolon, bruger et komma til adskillelse af teksten og variabel 1, fremkommer der kun den blinkende indtastningscursor på skærmen. Hvis man bruger semikolon, fremkommer der foruden teksten tillige et spørgsmålstegn bag den. Hvis flere variabler indlæses med et input, så skal også indtastningerne - ligesom variabelnavnene - adskilles af kommaer. I det første program med INPUT-sætningen spørges f.eks. med en sætning efter et kort heltal p% og en streng o$.

REM Postleit
'P1.5.3-1

Først dimensionerer vi et felt post$ med 1000 strengelementer. Med INPUT bliver man spurgt om indtil 999 forskellige postnumre med tilhørende byer. De enkelte byer gemmes under feltvariablen post$. Når man er færdig med indtastningen, skriver man 0,0. Derefter kan man få fat i den tilsvarende by ved at indtaste postnummeret. Finder programmet herunder en tom streng, ved det, at der ikke er indtastet noget navn

Felter kan også have flere dimensioner. Det er tilladt med indtil 255 dimensioner. Hver dimension kan have indtil 32767 elementer. Man bør prøve at forestille sig dette antal billedligt. Den menneskelige hjerne kan kun klare tre dimensioner, men Amigaen har ikke sådanne begrænsninger. I det følgende er en skematisk fremstilling af et todimensionalt felt. Det kaldes også en tabel, fordi man kan forestille sig de to dimensioner som spalter og linier:

c(0,0)	c(0,1)	c(0.2)	c(0,3)	c(0,4)	c(0,5)	c(0,6)	c(0,7)
c(1,0)	c(1,1)	c(1,2)	c(1,3)	c(1,4)	c(1,5)	c(1,6)	c(1,7)
c(2,0)	c(2,1)	c(2,2)	c(2,3)	c(2,4)	c(2,5)	c(2,6)	c(2,7)
c(3,0)	c(3,1)	c(3,2)	c(3,3)	c(3,4)	c(3,5)	c(3,6)	c(3,7)

Tabel 1.2: En todimensional feltvariabel

I næste program anvendes med det samme et todimensionalt felt. Da det er besværligt at indtaste så mange værdier med "håndkraft", lader vi dem beregne. Feltet tre% indeholder de arealer, som retvinklede trekanter, med højder og bredder mellem 1 og 20, har. Feltets første indeks repræsenterer trekantens grundlinie og den anden højden.

REM Dreiecksflaeche
‘ P1.5..3-2

Efter dimensioneringen beregnes de mulige trekantsarealer i de to FOR-NEXT løkker, der ligger inden i hinanden. Man kan så indtaste grundlinie og højde, og programmet angiver de hertil hørende arealer. Ved denne forespørgsel beregnes arealerne ikke, men der ses efter i det todimensionale felt. I øvrigt stopper programmet ikke af sig selv, men skal afbrydes med menuordren STOP.

Som sidste eksempel vover vi os ind i et tredimensionalt felt. Et sådant felt kan man bedst forestille sig som en kasse. De to første dimensioner udgør grundfladen og den tredje højden af kassen. Da det stadig er ret let at forestille sig, beregner vi indholdet af alle mulige kasser med kantlængder mellem 1 og 20 og lægger resultatet ind i et tredimensionalt felt:

REM Quader
’P1.5.3-3

Det ses, at programmet ikke er mere kompliceret end eksemplet med arealberegning. Her beregner vi kassernes rumfang efter dimensionering i de FOR-NEXT løkker, der ligger inden i hinanden. Man kan angive værdierne af længde, bredde og højde, hvorefter rumfanget af kassen fremkommer på skærmen. Jeg håber, det fremgår heraf, at det også er let at programmere flerdimensionale felter.

1.6 Basic 1*1

I afslutningen af dette afsnit om de grundlæggende egenskaber ved Basic, skal vi lave lidt matematik med Basic. Vi har strejfet området tidligere i bogen. Her går vi noget dybere.

1.6.1 Over stikker under

Når man regner med Basic, kan man benytte de regneoperationer, man kender fra matematikken. Når man skal udføre en beregning, må man kende den rækkefølge, fortolkeren udfører de aritmetiske operationer i. Den følgende opstilling sammenfatter de vigtigste operatorer i det matematiske hierarki. Heri har potensopløftning den højeste og addition den laveste prioritet.

Operator	Regnemåde	Eksempel
^	Potensopløftning	V^4
-	Negation (negativt fortegn)	-V
* og /	Multiplikation og division med flydende komma	V*8 og V/34
	Heltalsdivision	V4
+ og -	Addition og subtraktion	V+7 og V-9

(logiske operatorer forklares i Kapitel 9)

Tabel 1.3 Matematiske operatorer

Følger flere operatorer på samme hierarkiske niveau efter hinanden, så udføres de fra venstre mod højre. Denne regel kan man ophæve ved at benytte parenteser.
Når der er flere parenteser inden i hinanden, udføres indholdet af den inderste altid først.

Eksempelvis:

Hvis man vil opløfte summen af 3 og 2 til 5. potens, er følgende fremgangsmåde forkert:

```
PRINT 3+2^5
35
```

På grund af operatorernes prioritering beregnes nemlig først 2^5=32 og 3 adderes så. For at nå det rigtige resultat, er det nødvendigt at bruge parenteser:

```
PRINT (3+2)^5
3125
```

Hvis man vil dele værdien 17 med resultatet af subtraktionen 11-5, er det lige så forkert at skrive således:

```
PRINT 17/11-5
-3.454545
```

Så divideres 17 nemlig først med 11 og der trækkes 5 fra. Også her giver parenteser det rigtige resultat:

```
PRINT 17/(11-5)
2.833333
```

Hidtil har vi kun anvendt operatorer på numeriske værdier; men der findes også operatorer til strengværdier. Den hyppigste er sammenkædningen af to eller flere strenge, der kan nås med plustegnet (+). Her er et eksempel:

```
a$ = "uper":b$ = "computeren ":c$ = "Amiga"
PRINT "S" + a$ + b$ + c$
```

Resultat: Supercomputeren Amiga

1.6.2 Logiske operatorer

betingelser (sandhedsværdier). Bag IF skal der f.eks altid følge en betingelse, der gør det muligt for fortolkeren at afgøre, hvor den skal fortsætte programmet: bag THEN eller bag ELSE, resp. END IF. I princippet er ganske vist ethvert numerisk udtryk eller en numerisk variabel en betingelse (en værdi på nul gælder som FALSK og en værdi på ikke-nul som SAND). Til formulering af sådanne betingelser bør man kende de vigtigste logiske operatorer og sammenligningsoperatorer. I dette afsnit præsenterer vi de logiske operatorer og i det næste sammenligningsoperatorerne.

Operator	Operand1	Operand2	Resultat
NOT	sand	-	falsk
AND	sand	sand	sand
	sand	falsk	falsk
	falsk	sand	falsk
	falsk	falsk	falsk
OR	sand	sand	sand
	sand	falsk	sand
	falsk	sand	sand
	falsk	falsk	falsk
XOR	sand	sand	sand
	sand	falsk	sand
	falsk	sand	sand
	falsk	falsk	falsk

Tabel 1.4 Logiske Operatorer

Med logiske operatorer kan man styre programudførelsen med IF-sætninger. Prøv først med NOT:

```
a=5:IF NOT 6 THEN PRINT "FALSK"  Resultat: FALSK
a=5:IF NOT 5 THEN PRINT "SAND"   Resultat: SAND
```

De hyppigste sammenligninger udføres med AND og OR. Det følgende program benytter AND i starten af IF-blokken og viser under programforløbet resultatet i henhold til operatoren AND.

REM und
'P1.6.2-1

Det følgende program anvender operatoren OR:

REM oder
'P1.6.2-2

Med programmet kan man udlæse operatoren OR's sandhedstabel. Man vil altid bruge OR, når kun én af to betingelser skal være opfyldt:

```
IF laveste% < 20 OR hoejeste% > 100 THEN PRINT "forkert indtastning"
```

1.6.3 Sammenlignings-operatorer

Til slut ser vi på de simplere sammenligningsoperatorer, som sammenligner to udtryk med hinanden med hensyn til lighed, ulighed, større, mindre osv. Resultatet af en sådan sammenligning er så enten sand (værdi -1) eller falsk (værdi 0). I almindelighed bruges sammenligninger i f.eks. IF-THEN eller WHILE-WEND sætninger. Den følgende tabel viser alle sammenligningsoperatorerne:

Operator	Betydning	Eksempel
=	lig med	værdi1 = værdi2
<	mindre end	værdi1 < værdi2
>	større end	værdi1 > værdi2
< > eller > <	ulig med	værdi1 < > værdi2
< = eller = <	mindre end eller lig med	værdi1 = < værdi2
> = eller = >	lig med eller større end	værdi1 = > værdi 2

Tabel 1.5: Sammenligningsoperatorer

For at forstå det bedre, bør man, i direkte funktionsmåde, afprøve nogle eksempler:

Indtastning	Resultat
print 27-12=15	-1 (rigtigt)
print 25> <12	-1 (rigtigt)
print 30< >30	0 (forkert)
a=5:b=7:print a>b	0 (forkert)
c=112:ca=14:print c> =ca	-1 (rigtigt)
a=5:if a then ?"sand"else ?"falsk	sand
a=0:if a then?"sand"else ?"falsk"	falsk

Tabel 1.6 Sammenligningsoperatorer i direkte funktionsmåde

Ligesom med numeriske værdier, kan man også anvende tegnstrenge sammen med sammenligningsoperatorer. Herunder sammenlignes altid to strenge, tegn for tegn, via ASCII koden:

Indtastning	Resultat
print "a">"b"	0 (forkert)
print "D"<"H"	-1 (rigtigt)
print "Pater"<"Peter"	1 (rigtigt)

Tabel 1.7: Sammenligningsoperatorer med tegnstrenge

Til slut i dette kapitel skal vi have et lille program efter så megen teori. Det demonstrerer anvendelsen af sammenligningsoperatorer. Da det anvender en kommando, vi først skal lære at kende i næste kapitel, har jeg anbragt det sidst. Prøv det, og glæd dig over resultatet.

```
REM Vergleich
REM P0.6.3-1 Vergleichsoperatoren
```

2 GRAFIK

- Med et tegneprogram kan man fremstille en tegning til indbygning i et Basic-program. Her begynder det at blive svært, for bortset fra at se på billedet, kan man ikke rigtig gøre noget ved det. Det ville ellers være rart at kunne ændre og tilpasse tegningerne efter behov. Det skal vi snart få lært. I dette kapitel skal vi lære kommandoer og sætninger, til tegning af grafik-billeder, med små og lidt større programmer. Det er ikke meningen, at læseren blot skal sidde stift og indtaste listninger. Tværtimod vil vi give flest mulige ideer til egne progammer. Derfor bliver der hele tiden brugt nye programeksempler til forklaring af de enkelte kommandoer, selv om det er lettere at udvide tidligere programmer. Men nu i gang.

2.1 Den første grafik-kommando

LINE STEP (x1,y1)-STEP(x2,y2),FARVE,bf
tegner en linie eller et rektangel på skærmen

LINE tegner en linie på skærmen fra punkt (x1,y1) til punkt (x2,y2). Option'en farve er den, linien skal have. Anvender man til slut option'en bf, bliver det rektangel, hvis diagonal udgøres af linien, udfyldt med den aktuelle farve. Punkterne angives normalt som absolutte positioner (koordinater) i parenteser; hvis der skrives STEP foran parenteserne, bliver tallene ikke betragtet som absolutte positioner, men som det søgte punkts afstand fra grafik-cursorens aktuelle position.

Denne alsidige kommando har læseren sikkert allerede afprøvet. Amiga-Basic manualen påstår, at option'en "farve" kun kan have de fire grundfarver. Det passer ikke; vi skal senere tegne med 32 farver ved hjælp af LINE. Først afprøver vi Amigaens hastighed med et lille linietegningsprogram.

REM linie
'P2.1-1 LINE, GOTO

Aktiver AmigaBasic med et dobbelttryk på venstre musetast, og indtast programmet. Bemærk, at det ikke standser under kørslen. Når den sidste linie er nået, hopper det nemlig tilbage til den første. Man skal derfor stoppe med Menu-kommandoen Stop fra Run-menuen, såsnart billedet er færdigt.

Med de tre LINE-sætninger har vi benyttet kommandoen på tre forskellige måder. Den første LINE-sætning tegner en skrå linie fra øvre venstre hjørne af

skærmen til midten af højre vindueskant. Variablen x får en ny værdi ved hvert gennemløb af programmet, og forskyder således de enkelte linier lidt. Den anden LINE-sætning fortsætter den første linie ved dens slutpunkt. Ved hver sætning er de absolutte skærmbilledkoordinater angivet. I den sidste LINE-sætning er det anderledes. På grund af STEP bliver grafik-cursorens aktuelle position til startposition. Linien tegnes altså nul billedpunkter i x-retningen og minus 186 pixels i y-retningen. (Bevægelser i x-retningen har et positivt fortegn til højre og et negativt til venstre. På y-aksen har bevægelser nedad positivt fortegn og bevægelser opad negativt.)

Prøv nu, hvordan LINE-kommandoen opfører sig ved tegningen af rektangler:

REM Tunnel
REM P2.1-2 LINE, IF THEN... ELSE

I fjerde linie får variablen f værdien 3. Her står f for farve. Hvis farverne ikke er ændret i Workbench'ens preferences, har de fire standardfarver følgende betydning:

- 0 = blå (baggrund)
- 1 = hvid (forgrund)
- 2 = sort
- 3 = rød/orange

Under hopmærket *auf modtager de to variabler for rektangelstørrelsen deres startværdier. Programmet går derefter til hopmærket tunnel. Den første LINE-kommando sætter et udfyldt kvadrat i midten af billedskærmen. De to første tal bag LINE-kommandoen er værdierne for rektanglets øverste venstre hjørne (200 billedpunkter til højre for det øverste venstre hjørne af skærmen og 95 billedpunkter nedad fra det øverste venstre hjørne). De to andre tal i den anden parentes er værdierne for rektanglets nederste højre kant. Derpå følger variablen f for rektanglets farve og option'en bf, som sørger for, at der tegnes et rektangel og ikke en linie. Hvis der kun angives b, tegner Amigaen omridset af rektanglet, f'et udfylder med farve.

Sidst i programmet kommer to IF-sætninger. I den første sætning afprøves, om rektanglet i mellemtiden er blevet så stort, at det vil overskride skærmkanterne. I så tilfælde forgrener THEN til hopmærket *farbe. Den anden IF-kommando har ansvaret for farveskift:

Hvis farve = rød så farve = hvid ellers farve = rød

IF f = 3 THEN f = 1 ELSE f = 3

Nu lader vi teorien ligge. Gå med musen til Run-menuen og vælg Start.

Det er måske for lidt, at der kun er fire farver til rådighed på en supercomputer med 4096 mulige farver? Før vi kan gøre noget ved det, må vi beskæftige os mere med skærmbilledet.

2.2 Åbning og lukning af vinduer

Man lærer hurtigt, hvad vinduer betyder i Amigaen. Før vi åbner et nyt, bør vi se på, hvor stort det vindue er, som vi har åbnet vore første programmer i. Hvis du tror, at det er 640 pixels i vandret retning og 200 pixels lodret, tager du fejl. Prøv at lade Amigaen søge efter det rigtige svar.

REM Bildschirm
REM P2.2-1 ermitteln der Bildschirmgroesse

Det vigtigste i programmet er de to følgende kommandoer:

> **PRESET STEP (x,y),farve**
> **PSET STEP (x,y),farve**
> sætter et billedpunkt på en bestemt skærmposition

x og y er de absolutte, eller med STEP de relative, koordinater til punktet. Angives ikke nogen farver ved PRESET, anvender Basic skærmens baggrundsfarve. På den måde kan denne sætning bruges til at slette linier:

```
FOR i = 100 TO 200: PSET(i,50) :NEXT i
FOR i = 200 TO 100 STEP-1 : PRESET(i,50) :NEXT i
```

Eksemplet tegner en line og sletter den straks igen.

Programmet P2.2-1 sætter henholdsvis en lodret og en vandret hvid linie ved den højre resp. øverste kant af vinduet. Ved siden af de to linier bliver der tegnet et smalt rødt rektangel, hvis sidelængde ikke mere kan være på skærmen. Ser man nøje efter, vil man bemærke, at den hvide line altid er en pixel længere, end det tilsvarende rektangel er højt eller bredt.

Programmet viser, at der maksimalt kan vises 617 punkter vandret og 187 punkter lodret i udlæsnings-vinduet. Det er imidlertid for lidt. Vi prøver at forøge det med nogle pixels; men dertil er det nødvendigt med lidt viden om SCREEN- og WINDOW-sætningerne:

SCREEN nummer, bredde, højde, dybde, mode
åbner en ny billedskærm med bestemte egenskaber

SCREEN CLOSE nummer
lukker billedskærmen med det angivne nummer

Værdien af nummer skal være mellem én og fire, og er betegnelsen for den nye skærm. Denne betegnelse skal også anvendes i WINDOW-sætningen for at fastlægge, hvilken billedskærm et vindue skal ligge i, når det første gang kommer frem. Værdierne af højde og bredde og mode afhænger af hinanden, og kan ikke anvendes efter forgodtbefindende. Værdien af dybde fastlægger antallet af bitplaner og dermed det antal farver, der kan vises:

Dybde	Farver der kan vises
1	2
2	4
3	8
4	16
5	32

Tabel 2.1: Bitplaner og farver

Værdien af mode fastlægger billedskærmens opløsning

Mode	Opløsning
1	320 pixels/linie med 256 (200) linier (uden interlace)
2	640 pixels/linie med 256 (200) linier (uden interlace)
3	320 pixels/linie med 512 (400) linier (med interlace)
4	640 pixels/linie med 512 (400) linier (med interlace)

Tabel 2.2: Mulige opløsninger for en ny billedskærm. Værdierne i klammer gælder for version 1.1 af operativsystemet.

Interlace-mode fordobler antallet af linier, men til gengæld tegnes kun halvt så mange nye billeder pr. sekund. Det medfører en meget tydelig, yderst generende flimren af billedet. Men nu modstykket til SCREEN:

WINDOW nummer,titel,(x1,y1)-(x2,y2),type,Screen
åbner et nyt vindue med bestemte egenskaber

WINDOW OUTPUT nummer
sender alt output fra et Basic-program til vinduet med det angivne nummer

WINDOW CLOSE nummer
lukker vinduet med det angivne nummer (fjerner det fra skærmen)

Den første kommandoform frembringer et nyt vindue, og viser det på skærmen. Nummer (mellem 1 og 4) giver det nye vindue en betegnelse, som det senere kan kaldes under. Hvis man kun anvender WINDOW med nummer og udelader resten af angivelserne, aktiverer det et allerede frembragt vindue og bringer det forrest. (x1,y1)-(x2,y2) angiver det øverste venstre og nederste højre hjørne af det nye vindue, og SCREEN bestemmer, i hvilken skærm det dukker op (se SCREEN). Det er klart, at et vindues mål altid skal være lidt mindre end skærmen. Type fastlægger, hvordan man kan manipulere med vinduet ved hjælp af musen. Det skal senere illustreres med eksempler.

WINDOW OUTPUT aktiverer et vindue, men bringer det ikke forrest. (Udlæsningen af grafik kan, med andre ord, også ske i baggrunden.) WINDOW CLOSE sletter vinduet med den angivne betegnelse.

Vi prøver vor nyerhvervede viden på et program til at konstatere skærmstørrelsen. I SCREEN-sætningen tager vi 2 som dybde-værdi (maksimalt fire farver) og ligeledes 2 som mode-værdi. Mode 2 betyder visning af 640 pixels pr. linie og 256 linier. Til WINDOW-sætningen bruger vi 2 som betegnelse, da betegnelsen 1 allerede er brugt til Basic-outputvinduet. Vi udelader titel, og type sætter vi til tallet 0. Skærmkoordinaterne udelader vi også. Derfor fremkommer vinduet med maksimal størrelse. Det reviderede program ser ud som følger:

REM Bildschirm 2
'2.2-2 SCREEN WINDOW

Programmet virker ikke med operativsystemets version 1.1. Bruger man den første version, skal alle y-værdier, der er større end 0, reduceres med 56. Den følgende linie, indtastet i direkte funktionsmåde, konstaterer den maksimale højde af skærmen:

PRINT PEEKW (PEEKL (WINDOW(7) + 46) + 14)

Det ville sprænge bogens rammer at forklare den sætning; det ville kræve en større fordybelse i Amiga-softwaren.

Hvis man lader dette program køre, vender man tilbage til output-vinduet med et tryk på en vilkårlig tast. Det sørger anvendelsen af følgende sætning, i slutningen af programmet, for.

> **INKEY$**
> afprøver, om et tegn er blevet indtastet og returnerer det som resultat

Hvis tastaturbufferen er tom, overgives en tom tekst (""). Programmet konstaterer det, og begynder forfra, så længe der trykkes på en tast.

At man virkelig kan fremkalde nye skærme, der er uafhængige af Workbench, ved hjælp af SCREEN, viser det følgende program:

REM ZweiSCREENS
'P2.2-3 FRE SPC TAB PTAB

Før vi ser nærmere på programmet, skriver vi det først af og starter det. Der fremkommer et vindue, der ligger omtrent midt på skærmen. I vinduet er der et rektangel, der hele tiden skifter farve. Ved den øverste kant af skærmen ses en hvid stribe. Kør med musens pil hen på denne hvide bjælke, og tryk på venstre museknap. Kør nu med nedtrykket museknap til midten af skærmen, og der fremkommer endnu et vindue bagved, hvor der også er et rektangel, der skifter farver. Vi ser nu to SCREENS samtidig på billedskærmen. Men også denne billedskærm har en hvid bjælke foroven. Gentag processen fra før, og træk billedskærmen en tredjedel ned. Bagved ligger endnu en skærm, nemlig Workbench'ens SCREEN. Den viser i øjeblikket Basic-vinduet skjult. Når man lukker den, ser man Workbench'en med alle sine ikoner. For at forlade programmet igen, behøver vi kun at trykke på én tast. (Hertil anvendes det allerede præsenterede INKEY.)

I den første linie af programmet, efter bemærkninger (REMs), bliver den disponible hukommelse i variablerne registreret:

> **v=FRE(x)**
> giver alt efter x udsagn om fri hukommelse

Ved FRE drejer det sig om en funktion, som leverer information (tal eller tekster). I dette tilfælde om fri hukommelsesplads.

x=1	disponibel system-hukommelse
x=2	disponibel AmigaBasic stak-hukommelse
x>-1	disponibel Amiga-Basic programhukommelse

I programmet *zwei SCREENS-to skærme spørger vi først om den disponible systemhukommelse. Derefter definerer vi to skærme med en opløsning på 320*200 pixels og en dybde på 5. Der frembringes to vinduer, et for hver skærm. I vindue 3 vises teksten, om den disponible hukommelse, før og efter aktiveringen af SCREENS og WINDOWS. Den egentlige programudførelse begynder ved labelen start. Vindue 2 aktiveres med WINDOW OUtPUT 2.Den efterfølgende LINE-kommando virker usynligt i baggrunden. Hvis man sletter OUTPUT-delen af kommandoen, skiftes der løbende om mellem de to SCREENS. Prøv det bare. Linien WINDOW 3 sørger for, at vindue 3 bliver forrest, og det farvede rektangel ses der. Før GOTO start afslutter løkken, undersøges det atter, om der er blevet trykket på en tast. Hvis det var tilfældet, lukkes de to skærme igen efter hinanden med SCREEN CLOSE.

Lad os vende tilbage til programeksemplets tekstudlæsning. Måske har læseren bemærket noget? Der er i alt fald meget at lægge mærke til. For det første det enorme hukommelsesbehov for et så lille program. Det kræver 86 Kbytes alene fra system-hukommelsen. Pas altså på med at eksperimentere med SCREEN. Det er let at få Amigaen til at crashe. Hvis man f.eks. vil anbringe en opløsning på 640*200 pixels i SCREEN-sætningen, crasher programmet. Så er der kun én ting at gøre: ud med disketten og start på ny.

I programeksemplet er der tre forskelllige kommandoer til anvendelse af tabulatoren:

SPC(n)
udlæser n tomme mellemrum

Denne kommando giver mellemrum i en tekstlinie:

PRINT "Amiga"SPC(5)"Basic"
giver i skærmbillede, på printer og i sekventiel fil
Amiga   Basic

TAB(n)
sætter tekst-cursoren på positionen n pladser til højre for vinduets venstre kant (den lodrette position uændret)

Med kommandoen kan man få en formatteret udlæsning:

```
PRINT "Computer"TAB(25) "Side 20"
PRINT "COMPUTER-Tilbehør"TAB(25) "Side 22"
PRINT "Printer"TAB(25) "Side 25"
```

giver:

Computer	Side 20
COMPUTER-tilbehør	Side 22
Printer	Side 25

Den sidste tabulatorfunktion sætter en grafisk tabulator:

PRINT PTAB(n)
sætter tekst-cursoren på positionen n pixels til højre for vinduets venstre kant (den lodrette position uændret)

Denne kommandos funktion ses let i vort sidste program. Under ordet *nachher finder man tallet for den resterende hukommelsesplads. Tallene står ikke direkte under bogstaverne, men lidt forskudt derunder.

Før vi går til næste kapitel, kan læseren roligt lege lidt med SCREEN og WINDOW. Men pas på hukommelsesbehovet. Forkerte angivelser er ofte årsag til crash.

2.3 Hvorhen med programmerne?

Når man har skrevet et par programeksempler, vil man sikkert gerne gemme dem; men ser man nærmere på Amiga-Basic disketten, bemærkes at den er godt optaget. Som man kan få at vide med Info fra Workbench-menuen, er der allerede brugt 1515 af i alt 1758 blokke. Med 512 bytes pr. blok giver det 124 Kbytes til overs til egne programmer. Tror man, at det er en hel del, kender man Amigaen dårligt. De fleste specielle Amiga-egenskaber kræver enorme mængder lagerplads. De resterende 124 Kbytes er lige nok til at gemme nogle små programmer og tre (3) grafikbilleder!

Selvfølgelig kan man gemme sine Basic-programmer på en vilkårlig diskette med den rette fil-specifikation, og loade den igen. Men for det første er det temmelig omstændeligt altid at opgive hele rutenavnet og diskettestationen, og for det andet vil vi jo udnytte alle fordelene ved pull-down menuer. Det er heller ikke til megen hjælp at slette Amiga Tutor'en på Basic-disketten, men prøv bare. Derfor skal vi nu se, hvordan man kan fremstille en programmeringsdiskette med plads nok til sine egne programudviklinger.

Anvend aldrig original-disketten til de disketteoperationer, der beskrives i det følgende. Den bør kopieres og opbevares et sikkert sted. Sker en grov fejl under programmeringen, eller nogen snubler over computerens netledning,

så kan det være uigenkaldeligt forbi med dataene. Det er sjældent, der sker sådan noget. Men når det sker, bør det ske med en kopi af original-disketten.

Tilbage til Amiga-Basic disketten. Grunden til at det ikke nytter at slette Tutor, ligger i alle de underprogrammer og hjælpefiler, der hører til den. Man kunne slette dem alle ved hjælp af KILL-kommandoer. Vi vælger hellere en ny diskette.

Tag en tom diskette, eller en hvis indhold er overflødigt. Læg den i og vent til dens ikon (eventuelt navnet DF0:BAD) ses, og klik to gange på det. Vælg nu Initialize-Disk fra Disc-menuen. (Computeren beder måske om Workbench-disketten.) Når det røde lys er slukket, tager man den ud, og lægger den nye diskette i igen. Amigaen spørger, om man er sikker på, at man vil initialisere (en så farlig kommando skal siges to gange). Klik nu på Continue.

Når processen er forbi, åbnes disketten med et dobbeltklik på ikonen, og man ser en diskette, der er helt tom bortset fra papirkurven. Ombenævn nu disketten (med Rename) til BFA. Lad vinduet være åbent og indlæg kopien af Amiga-Basic disketten. Åbn nu også dette diskettevindue og transporter Amiga-Basic ikonen over i den nye diskette med musen. Når man flytter en ikon fra en diskette til en anden, kopierer Amigaen automatisk dens indhold, men sletter det ikke på originaldisketten. Man har altså stadig Amiga-Basic på diskette nr. 1. Nu opfordrer computeren til nogle disketteskift, indtil programmet er kopieret over på den nye diskette.

Den samme procedure gentages straks med Basic-Demo skuffen. Man kan også godt tømme Demo-skuffen på BFA-disketten, men lad spriteeditoren og de to bilioteksrutiner blive der. Prøver man nu den nye BFA-diskette med Info, ser man, at der er 1513 blokke, dvs. 774 Kbytes til rådighed. Tag straks en kopi af den nye diskette til næste gang, der er brug for en Basic-diskette. Sådanne kopier skal have andre navne.

Efter kopieringen ser vi på dette lille program, der viser den store effekt, man kan opnå med LINE-kommandoen og lidt omhu.

REM Kelch
'P2.3-1 COS ATN WRITE SWAP

Programmet tegner to bægerformede liniebundter og viser hermed anvendelsen af nogle nye funktioner og kommandoer. For eksempel blev liniernes vinkler tegnet med funktionerne COS og ATN:

v=COS(x)
giver cosinus til værdien x (vinkel angivet i radian)

Værdien x skal først omregnes til radian. Den nødvendige beregning for en vinkel på 43 grader er:

v=COS (43*pi/180)

Indsættes 3.141593 som værdi af pi, fås som resultat af COS-funktionen 0.7313536

v=ATN(x)
beregner arctangens til x

Denne funktion giver længden af cirkelbuen i enhedscirkelen med r=1. For at omregne det til vinkelgrader skriver man

vinkel=180/pi*ATN(x)

Hvis v=3.14 fås: vinkel=180/3.14*3.14=180 grader.

Her er begge funktioner anvendt uden særlig matematisk baggrund. De viser kun, hvordan matematiske funktioner kan give grafiske effekter. To SWAP-sætninger sørger under programudførelsen for, at hver linie tegnes i en ny farve.

SWAP variabel1,variabel2
ombytter værdierne af to variabler af samme type

En streng-variabel kan altså ikke ombyttes med en numerisk variabel. I slutningen af kapitelet er der input-rutine, der demonstrerer en mulig anvendelse af SWAP. Til *kelch programmet anvendte vi en ny sætning:

WRITE liste af udtryk
viser værdierne af udtryk i det aktuelle vindue, ligesom PRINT ville vise listen i output-vinduet.

Udtrykkene skal adskilles af kommaer eller semikoloner. Modsat PRINT-kommandoen udlæses også strenges anførelsestegn. Ligeledes har komma og semikolon ingen formatterende funktion her og behandles ens.

2.4 Mere farve

Her begynder vi første kapitels program-projekt. Vi går straks i gang med farverne ved hjælp af PALETTE-sætningen:

> **PALETTE farvebetegnelse,rød,grøn,blå**
> definerer for farvebetegnelse en farvetone ud fra de tre grundfarver rød, grøn og blå

Farvebetegnelse skal være et tal mellem 0 og 31, og svarer til en af de 32 farver, Amigaen kan vise på skærmen samtidig. Angivelserne for rød, grøn og blå bestemmer den pågældende farves bidrag til den resulterende farvetone. Hvis angivelserne for rød, grøn og blå alle er 1.0, giver det lysende hvidt. Er de alle nul, giver det sort. Der er 16 mulige værdier mellem 0.00 og 1.00 for hver bestanddel. Det giver 16*16*16 = 4096 forskellige farvetoner. Hvorfor lige 16? Fordi der netop er en halv byte (en nibble) til rådighed, og 4 bits kan repræsentere 16 forskellige intensiteter.

Hvis man virkelig vil programmere med 32 farver og ikke er tilfreds med de forudindstillede, skal man indtaste 32 PALETTE-kommandoer. Først anlægger vi dog en tom skuffe med navnet underprogrammer på BFA-disketten.

REM .2.4-1
Farbtafel

De her forudindstillede farver er måske ikke det bedste opnåelige med 32 farver, men man bør bibeholde dem til vort program-projekt, for at undgå misforståelser. Ved konstruktion af egne programmer kan de være udgangspunkt for fremstilling af egne farver. Bemærk, at man ikke bør anvende vilkårlige decimaltal til farvekomponenterne. Man skal anvende følgende værdier til de 16 mulige intensiteter:

```
0.00 0.07 0.13 0.20 0.27 0.33 0.40 0.47
0.53 0.60 0.67 0.73 0.80 0.87 0.93 1.00
```

Gem nu programmet med følgende kommando i direkte funktionsmåde (bemærk option a):

```
save "Underprogram/Farvetabel",a
```

Nu er det på tide at se farverne. Slet programmet *Farbpalette, der ikke kan køre alene, og indtast følgende program:

REM Farbpalette - Darstellung von 32 Farben
'PP2.4-2

Subrutinen *Taste venter på et tastanslag, og er alternativ til den enkelt-linie, der er forekommet i tidligere programmer. I FOR-løkken ovenfor tegnes 32 rektangler i forskellige farver.

Forfra i programmet opretter man en ny skærm med dybden 5 - altså 32 farver - med SCREEN. Så frembringes et nyt vindue i denne SCREEN. Vinduestitlen er en instruktion til brugeren. Det er nemt at benytte den type meddelelser, og så skal vinduet ikke selv udlæse nogen tekst. Man kan ændre billedet, ja slette vinduet, og alligevel forbliver oplysningen synlig.

Efter WINDOW kommer et hop til en subrutine kaldet *Farbtafel, der endnu ikke er i programmet. Det afhjælper vi ved i direkte funktionsmåde at indtaste:

merge "BFA:Underprogram/Farvetabel"

Straks ligger underprogrammet efter hovedprogrammet. Hvis ikke, skyldes det sikkert, at programmet ikke først er gemt med option A.

Merge Filangivelse
tilføjer et program (i ASCII-format) til et allerede bestående.

Programmet kan startes, og der er de 32 farver. Inden vi går videre, bør man kigge på de 32 standard-farver. Hertil inaktiveres linien GOSUB *Farbtafel med en apostrof og genstarter programmet.

2.5 Regnbuen

Før vi fortsætter, skal vi prøve at opbygge et billede med de 32 farver. Vi loader programmet *Farbtafel og ændrer det lidt:

- ombyt farverne 1 og 10
- slet RETURN i slutningen af programmet
- skriv i stedet GOTO *Anfang på samme sted

Gem så det ændrede program som "*Farbtafel1" med option A i ASCII-form. Så kan vi begynde. Det er bedst at indtaste listningen og gennemgå den punkt for punkt ved hjælp af teksten.

REM Regnbuen
REM P.2.5-1

Programmet er spækket med nye kommandoer. Linien DEFINT A-Z definerer alle variabler som korte heltal. SCREEN og WINDOW kender vi også; men så kommer et smart trick. Man har nok bemærket, at subrutinen *Farbtafel mangler i programmet, men alligevel kaldes. Vi behøver ikke at kalde denne fil i direkte funktionsmåde; vi lader programmet gøre det!

CHAIN MERGE filangivelse,linie,ALL,DELETE område
loader et andet Basic-program og starter det

Der er en række options til kommandoen, der ikke alle skal nævnes her. For programmet *Regenbogen betyder MERGE bagved CHAIN, at Basic-programmet *Farbtafel1 tilføjes fra skuffe-underprogrammmerne fra linie 10 og så startes. Derfor er sidste linie i programmet også 10 REM. Det er værdifuldt at kunne kalde et andet program med CHAIN og overgive værdier hertil. Vil man det, skal man benytte CHAIN således:

CHAIN "programnavn",,ALL

Med optionen ALL bevares værdierne af alle variabler, under kaldet af det nye program, og kan bruges heri. Hvis kun en del af variablerne skal videregives, skal de angives enkeltvis i begyndelsen af programmet:

COMMON Variabel1, Variabel 2,...
Bevare variablerne under en CHAIN-kommando

Kommandoens virkning vises på to små programmer:

REM Abloesung
'P.2.5-2

I programmet *Abloesung gives variablerne a-f forskellige værdier; men i COMMON-sætningen nævnte vi kun variablerne a-e. Husk at gemme programmet *Ausfuehren i ASCII-format. Når man starter programmet *Abloesung vises værdien af variablerne på skærmen. Efter en kort venteløkke loades programmet *Ausfuehren med CHAIN-kommandoen og herved slettes programmet *Abloesung. Der foretages og vises simple regnestykker med variablerne. Variablen f har nu værdien nul, selv om den tidligere havde en anden værdi i programmet *Abloesung.

Vi vender tilbage til programmet *Regenbogen. To linier bag CHAIN-kommandoen er en ny kommando:

COLOR forgrund,baggrund
fastlægger to af palettens farver som forgrunds- og bagggrundsfarve

Med sætningen kan man vælge enhver farve fra PALETTE til forgrund og/eller baggrund:

COLOR 10,25

Farve 10 bruges til skrift og grafik-kommandoer, hvor der ikke angives nogen farve. Farve 25 danner baggrund og bruges f.eks. også af PRESET-kommandoen til at slette punkter.

De to følgende linier tegner et rektangel, og hermed er løftet fra 1. kapitel opfyldt. Rektanglerne tegnes, som det ses, i farvetabellens farver 17 og 26 og ikke kun i farverne 0 - 3. Det samme gælder for CIRCLE-sætningen, som straks skal lave en regnbue med 32 farver.

I de første to WHILE-WEND løkker kaldes subrutinen *baum gentagne gange. Efter at de to rækker nåletræer er tegnet, konstueres der et hus, et par løvtræer og så kommer regnbuen.

CIRCLE STEP (x,y),r,farve,start,slut,ratio
Tegner en cirkel eller en ellipse om midtpunktet x,y med radius r

Når der kun angives (x,y),r og farve, skulle vises en cirkel. Med start og slut kan man angive, at der kun skal tegnes en del af hele cirklens 360 grader, nemlig buen mellem vinklerne start og slut, (som også skal angives i radian). Hvis man ikke vil have en cirkel men en ellipse, kan man angive med ratio, hvor stort forholdet mellem højde og bredde skal være. Variablen r gælder i så fald den største af de to radier (den vandrette eller den lodrette).

Regnbuen er sammensat af 32 ellipsestykker, der tegnes af følgende sætning. Kun angivelsen af centrum for ellipsen og farven kan ændres.

```
CIRCLE (160,a),  250,  i,  3,  2.9
x,y = 32*forskudt,  radius,  farve,  start,  slut
```

Subrutinen *baum tegner et træ og anvender to meget effektive grafik-kommandoer til det:

AREA STEP (x,y)
definerer en side af en mangekant (en polygon)

Den første AREA-kommando i et program fastlægger polygonens første hjørne; de følgende trækker linier til de opgivne punkter og føjer således nye kanter til polygonen. (Det bør helst fortsætte til polygonen er lukket, den sidste AREA-sætning bør altså føre tilbage til starten.)

AREAFILL,mode
lukker det polygon, der tidligerede blev defineret med AREA, og udfylder det med et mønster

Det er vigtigt, at koordinaterne i de enkelte AREA-sætninger fører til en lukket mangekant, ellers kan den ikke udfyldes. Hvis mode sættes til 1, inverterer Basic det indre af polygonen. Er ingen mode angivet, fyldes mangekanten med PALETTE-farve 1. (Mere præcist bliver polygonen udfyldt af et mønster; men hvis dette mønster ikke ændres med Pattern-kommandoen, er det en ensartet udfyldt flade.) Der er kun to farver til udfyldning af polygoner. Derfor måtte PALETTE-farverne 1 og 10 ombyttes, ellers var træerne blevet brune, groft realistisk!

Nu til den sidste nye kommando i regnbue-programmet. I subrutinen *lbaum tegnes og udfyldes en cirkel.

PAINT STEP(x,y),indrefarve,rand
udfylder en lukket flade, hvori punktet (x,y) ligger, med indrefarve, fladen begrænses af punkter med farven rand.

For kommandoen gælder, modsat manualens påstand, at fladen kan udfyldes med en vilkårlig farve ud af de 32. Men desværre kan fladen kun udfyldes med en farve, der danner rand. Hvis man har begrænset en flade med en anden farve, end den man udfylder den med, løber farven ud over hele billedet. Pas derfor på. Det nytter ikke, at man angiver samme farve for rand.

Før programmet startes, bør man gemme det i en skuffe med navnet grafikeksempler. Men nu noget lettere. Følgende program tegner nogle harer, der består af ellipser.

REM hasen
'P.2.5-4

Et par afsluttende tips om gemning. Når man skriver eller indtaster længere programmer, er det klogt at gemme dem af flere gange. Det går let med me-

nu-kommandoen Save. Det er kedeligt at tabe mange timers arbejde på grund af en f.eks strømsvigt, og det sker oftere, end man tror.

2.6 Dam

Her skal vi lære Amiga-Basics mest effektive grafik-kommandoer at kende: kommandoer, der forskubber hele dele af et billede. Det er smart at kunne sige: "Tag billedudsnittet i midten af billedet og læg det i nederste højre hjørne". Det sker let med GET og PUT skærmbilledområde.

> **GET(x1,y1)-(x2,y2),feltvariabel(indeks,...,indeks)**
> gemmer et rektangulært skærmbilledudsnit i et numerisk felt
> **PUT STEP (x,y),feltvariabel (indeks,...,indeks,)mode**
> lægger et gemt billede tilbage fra et numerisk felt til skærmen

Koordinat-angivelser gælder for GET, ligesom ved konstruktion af rektangler, det øverste venstre hjørne og det nederste højre hjørne af det rektangel, der skal gemmes. Derimod markerer værdierne x og y til PUT-kommandoen det øverste venstre hjørne af det område, hvori det gemte billede skal dukke op. Se engang på følgende eksempel:

REM Wald
'P.2.6-1 GET PUT

I programmet dimensioneres *Baum som et kort heltalsfelt. Mere præcist, man definerer ikke træet, men et rektangulært skærmbilledudsnit, som det ses af GET-sætningen. Hvor mange elementer feltet skal have, kan man beregne ud fra billedudsnittets mål og skærmens dybde efter følgende formel:

6+(y2-y1+1)*2*INT((x2-x1+16)/16)*dybde/2

Hvor (x1,y1) er øverste venstre hjørne, (x2,y2) nederste højre hjørne af det rektangel, der skal gemmes, og dybde antallet af bitplaner. I dette program med kun fire farver, er dybde lig 2. Fordi vi har dimensioneret som kort heltalsfelt
(et element=2 bytes), deles farvedybden med 2. Nu indsætter vi blot vort træs værdier i formlen. Det er lettere end det ser ud til.

6+(40-0+1)*2*INT((20-0+16)/16)*2/2 og vi får 167

Hvis feltvariablen havde været defineret som langt heltalsfelt, ville 83 elementer have været tilstrækkeligt.

Vi går videre i udførselsen af programmet *Wald. Først vælges baggrundsfarven brun og forgrundsfarven grøn med PALETTE-sætningen. Programmet tegner et simpelt træ med AREA og AREAFILL. Så går GET i aktion. billedområdet fra øverste venstre hjørne til koordinaterne x = 20 og y = 40 hentes til feltet bild%-*billed%. Med to FOR-NEXT løkker inden i hinanden sætter PUT så næsten 200 træer fra dette felt tilbage på skærmen. Det sker med en imponerende fart.

Det var en forsmag på, hvad GET og PUT kan. I næste program får vi en praktisk menu, og vi fremtrylle et damspil på skærmen. Det kræver jo bevægelige brikker, og hertil behøves Bobs eller Sprites, tror man måske. Nej, vi får ikke brug for disse objekter, som vi kommer ind på senere. Det hele klares med GET og PUT. Lad os se på det centrale: spilafviklingen

REM Dam
'P.2.6-2

Efter DEF-kommandoen fastlægges alle variabler som korte heltal. Så dimensioneres feltet *Stein til 71 elementer. Dambrikkerne er sorte og hvide, og da man næppe kan se sort på den forudindstillede mørkeblå bagggrund, indstiller vi baggrunden på lyseblåt med PALETTE-sætningen.

Så kommer en interessant kommando: ON MOUSE, der gør det muligt at sammenfatte hovedprogrammet til et kort programforløb.

> **ON MOUSE GOSUB hopmærke**
> hvis der trykkes på venstre musetast, forgrener programmet til detangivne hopmærke (som kan være et linienummer eller en label)
> **MOUSE ON**
> virkningen af ON MOUSE aktiveres
> **MOUSE OFF**
> inaktiverer ON MOUSE
> **MOUSE STOP**
> afbryder forbigående virkningen af ON MOUSE

Disse kommandoer hører sammen. Hvis man glemmer MOUSE ON efter ON MOUSE GOSUB, sker der slet ikke noget. Med MOUSE STOP bemærker fortolkeren, om venstre museknap er nedtrykket. Så snart der kommanderes MOUSE ON igen, forgrener programmet til den tilsvarende rutine. Efter musekommandoerne er der to WHILE-løkker inden i hinanden. De styrer to forskellige valgmuligheder. Den inderste løkke kan kun vækkes med venstre museknap (SLEEP). Den yderste WHILE-løkke aktiveres, så snart *ende ikke mere er sand, dvs. får værdien nul. Programmet gennemfører en ny start,

hvor værdien af *ende atter sættes til logisk sand (0). Straks når den ydre løkke forlades, afsluttes programmet med kommandoen END.

Lad os se på den første subrutine:

*** ein**

Efter linien hvor billedet blev slettet med CLS, er der en ny kommando: CALL. Den kalder noget, f.eks. et Basic-underprogram (ikke at forveksle med en subrutine, og tillige underprogrammer i maskinsprog eller biblioteksrutiner. Vi nøjes med at kalde et Basic-underprogram.

> **CALL hopmærke (argumentliste)**
> kalder et Basic-underprogram og overgiver det en liste af værdier (argumenter)

Forskellen på et underprogram og en subrutine? Basic administrerer et underprogram, næsten som var det et selvstændigt program. Underprogrammer styrer selv deres variabler, så det samme variabel-navn kan betegne forskellige variabler i hoved- og underprogram. Hvis en variabel i anvendes i hovedprogrammet og har værdien 4, så har et i, der anvendes i underprogrammet, ved starten af dette værdien 0. Tildeler man denne variabel en værdi i underprogrammet, så bevares den heri. Forlades underprogrammet, svarer Basic atter 4, når man indtaster PRINT i. De to variabler af samme navn i hoved-og underprogram, har nemlig intet med hinanden at gøre. Det giver mulighed for at anlægge et bibliotek med stadigt tilbagevendende rutiner, uden at skulle passe på de variabler, der anvendes i bilioteksrutiner, hvilket man skal med subrutiner.

Et underprogram, der kan kaldes med CALL, defineres således:

> **SUB navn (variabelliste)STATIC**
> Begyndelsen af et Basic-underprogram
> **END SUB**
> slutningen af et Basic-underprogram
> **EXIT SUB**
> forladelse af et Basic-underprogram før det afsluttes

I programmet Dam overgives ingen program-variabler til underprogrammet med CALL *Spielfeld; de behandles senere. Der følger seks kald til et underprogram med navnet start, hvorunder det får tre forskellige variabler med: br = bredde af venstre kant af dambrættet, farven sort eller hvid og *anf til anbringelsen af de enkelte brikker i y-retning. Til slut sætter vi teksten "ny" i et sort rektangel øverst til venstre i vinduet, og ved siden af det "slut" i et rødt

rektangel. Den sidste kommando i denne subrutine sætter igen variablen *ende på logisk sand. Kig nærmere på det første underprogram:

*SUB spielfeld STATIC

Variablen fa indeholder farven, som dambrættet vises i. Af grunde, vi senere skal omtale, er det ikke muligt at sætte brikkerne på to forskellige baggrundsfarver. For at spilleren skal kunne sætte brikkerne korrekt, tegner vi firkanter med hans brikkers aktuelle farve. Da et rektangel med normal stregtykkelse ikke virker særligt godt, tegner vi to inden i hinanden. Det sker med de to LINE-kommandoer, der står lige under hinanden. De to følgende FOR-løkker inden i hinanden udføres således: første række, fra venstre til højre: hvidt kvadrat, farveskift, sort kvadrat, farveskift osv., indtil der er tegnet otte. Så forlades den indre løkke, farven fa skiftes, så der nu begyndes med et sort kvadrat, og der gås tilbage til den indre løkke. Det hele sker otte gange for at tegne brættets otte linier.

Det andet underprogram sætter de to gange tolv brikker (hvide og sorte cirkler) på startpositionerne:

*SUB start (b,f,a) STATIC

Da alle nødvendige værdier medgives underprogrammet, bliver det meget kort. Brikkerne tegnes med CIRCLE-sætningen i hver andet felt ovenfra og nedefter. Da standard-værdien 1 for ratio i CIRCLE tegner ellipser, er værdien ændret til .52. Nu kan vi spille. Kast et blik på programstyringens kerne. Alt, hvad der kan gøres med musen, bearbejdes her:

*mauskontrolle:

Først spørges om musepositionen. Så kontrolleres om der er klikket i et af de to felter "ny" eller "slut". I første tilfælde nystartes programmet, i andet tilfælde afsluttes det, idet variablen *ende und/oder aus sættes til logisk falsk (0). Derved forlades WHILE-WEND løkken.

Er der hverken klikket på "ny" eller "slut", beregnes en korrigeret position, som brikken indlæses med (GET). Derpå sættes den på den samme position igen, og den forsvinder totalt.

Musepositionen og spørgsmålet, om der er trykket på musetasten, fastslås med funktionen MOUSE, der endnu ikke er forklaret

v = MOUSE(n)
leverer alt efter n forskellige informationer om musetastens tilstand og dens position

For at vide, om noget er ændret ved musens tilstand, aflæser man MOUSE(0). Hvis funktionen giver en værdi 0, er der sket en ændring af musens tilstand. De andre muligheder for n (1-16) vises, når de optræder i programmet.

WHILE-WEND løkken gennemløbes, sålænge museknappen holdes nede. Ved hvert gennemløb hentes den aktuelle position, sættes lig det øverste venstre hjørne af det indlæste (GET) billedudsnit, og det bringes igen på skærmen med PUT. Efter en kort pause slettes brikken med PUT. Fordi Amigaen udfører de fleste grafikordrer meget hurtigt, ville man slet kunne opfatte et så hurtigt skift mellem GET og PUT, derfor bremses programmet med en tidsløkke. Når den venstre musetast slippes, forlades WHILE-WEND løkken, og musens korrigerede position fastslås. På den korrigerede position sættes brikken på brættet. Til sidst kaldes underprogrammet *Spielfeld, for at genopbygge brættet. Kig nu på subrutinen *posholen, som fastslår musens position:

*posholen

Forrest står en tilsyneladende meningsløs funktion, som ikke skal bearbejdes i programmet, men er nødvendig, for at de to følgende funktionskald giver information. MOUSE(1) giver x-koordinaterne og MOUSE(2) y-koordinaterne på tidspunktet for det sidste MOUSE(0)-kald. Hvis man glemmer at kalde MOUSE(0) først, giver begge nul. Vi kan nu håndterer musen med Basic, og går til sidste subrutine i programmet:

*poskorr

Også her aflæses musepositionen og den bearbejdes i to formler. Her bliver en position, der ligger i et kvadrat på brættet, omregnet til koordinater, der ligger nøjagtig i midten af kvadratet.

For at opklare eventuelle andre spørgsmål om programmet indtaster og starter vi det, så snart det er gemt. Når brættet er færdigtegnet, kan man gribe aktivt ind i begivenhederne med musen. Klik på en brik og hold knappen nede. Brikken springer hen under musens pil og bliver liggende i det felt, hvor museknappen slippes. Det er ikke nødvendigt at klikke nøjagtigt midt på en brik for at kunne bevæge den. Musens pil skal bare være i det pågældende kvadrat. Programmet overtager den nøjagtige placering. Brikker, der er slået, kan lægges til venstre og højre for brættet. Ved afslutningen af et spil, klikker

man blot på feltet ny, hvis man vil fortsætte. Straks får man et nyt med brikker i udgangspositionen.

Måske har læseren prøvet at lægge en sort brik oven på en hvid? Ellers gør det. I stedet for en hvid eller sort brik får man nu en rød, der godt kan anvendes som en dam. Men hvis man lægger to brikker af samme farve oven på hinanden, forsvinder de begge. Vi ser nærmere på fænomenet.

Ansvaret har PUT-sætningens mode. Når man gennemser programmet, vil man opdage, at den aldrig bliver brugt. Ved de fleste kommandoer er der en forudindstillet standardværdi, der anvendes i mangel af andet: XOR. Desuden findes mode'rne PSET, PRESET, AND og OR. Vi interesserer os foreløbig kun for PSET og standardværdien XOR; kun de er anvendt i damspillet.

Under den eksklusive eller-sammenknytning, som XOR bevirker, sammenknyttes to sandhedsværdier (bits) som følger:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Bemærk at både to falske og to sande operander gir et falsk resultat.
Kun når de to operander er ulig hinanden, giver det et 1 (= sand). Det betyder, at to brikker af samme farve, der sættes oven på hinanden, forsvinder (da alle punkter bliver lig med nul). Det samme pricip blev benyttet til at flytte en brik. Herunder blev brikken, med den lille pause p.g.a. tidsløkken, sat to gange på samme sted. Brikken kommer så til syne og forsvinder atter for eventuelt at dukke op et andet sted.

2.7 Tordenvejr

Vi har lært at gemme grafik forbigående i en variabel med GET-kommandoen og sætte den på skærmen med PUT. Men hvordan opbevarer man billeder i længere tid? Det skal vi se med et programeksempel, der begynder med følgende linier:

REM Gewitter
REM P.2.7-1

Igen er der en ny kommando:

CLEAR,programlager,staklager
sletter alle variabler og fastlægger hukommelsesopdelingen på ny.

Når man kalder Amiga-Basic, får man visse data i output-vinduet. I den sidste linie står: 25 000 bytes free in Basic. Disse 25 000 bytes er reserveret til programmet. Til staklageret er kun 4789 bytes, men det er i reglen tilstrækkeligt. De 25 000 bytes til programmet er dog for lidt. I programmet *Gewitter gemmes hele skærmbilledet i hukommelsen med GET-sætningen. Det kræver alene 38 Kbytes. Da programmet også kræver plads, reserverer vi 45 000 bytes med CLEAR.

Screen 1, 320,200,5,1

Programmet styres ved tryk på taster. Hvilke taster, udlæses med PRINT og tillige permanent som vindues-titler. Kommandoerne bag hopmærket start skriver menuen på skærmen med PRINT, medens der bag hopmærket *auswahl ventes på tastaturindlæsning og forgrenes til de enkelte underprogrammer.

*auswahl

Subrutinen *einlesen beder først brugeren om et navn på det billede, der skal indlæses og vises. Teksten overgives til strengvariablen filename$. Hvis brugeren ikke indtaster et navn, men blot trykker return, forgrener programmet tilbage til *einlesen. Såsnart der foreligger et navn, forsøges det at indlæse den tilsvarende fil:

OPEN "filnavn" FOR mode AS #filnummer
åbner en fil til dataind- eller udlæsning

Mode kan være:

- OUTPUT til sekventielle udlæsning
- APPEND til tilføjelse til en sekventiel fil
- INPUT til sekventiel indlæsning

I eksempelet åbnes en fil med nummeret 1 til sekventiel indlæsning

Efter kald af subrutine *bilddim og udlæsning af en melding på skærmen følger en WHILE-løkke. Funktionen EOF er interrupt-melding for løkken.

v = EOF(filnummer)
leverer den logiske værdi sand, når slutningen af en sekventiel fil nås under læsningen

WHILE-løkken kører altså, til slutningen af dataene er nået. Derfor kan man indlæse filer med ubekendt længde. Linien inden i WHILE-løkken sørger for, at dataene indlæses og opbevares i elementer af *bild&. Hertil bruges to nye funktioner.

v$ = INPUT$(længde,#filnummer)
overgiver en tegnstreng af bestemt længde fra en sekventiel fil (ellertastaturet) til en strengvariabel

Så benyttes funktionen CVL til at forvandle en streng af længden 4 (altså 4 bogstaver = 4 bytes) til et langt heltal. CVL er blot en af en type funktioner, Basic bruger til lignende formål. Følgende funktioner forvandler strengindhold til numeriske værdier:

v = ASC(v$)	1-byte tegn (ASCII-kode) til decimalværdi
v = CVI(v$)	2-bytes tegnstreng til kort heltal
v = CVL(v$)	4-bytes tegnstreng til langt heltal
v = CVS(v$)	4-bytes tegnstreng til enkelt-præcisions tal
v = CVD(v$)	8-bytes tegnstreng til dobbelt-præcisions tal

Dataovertagelsen forekommer måske ret kompliceret. Fordi data (også tal) i en sekventiel fil altid lagres som ASCII-tegn, skal tal først indlæses som tegnstrenge og så forvandles. På grund af de fire repræsentationsmåder kræver det fire forskellige funktioner. Den femte funktion, der er anført først, er nødvendig, når man kun skal indlæse en enkelt byte.

PUT sætter det indlæste billede på skærmen og ERASE sletter det fra hukommelsen.

ERASE feltvariabel,feltvariabel
sletter angivne feltvariabler og frigør den nødvendige hukommelse

Subrutinen *einlesen er ansvarlig for indlæsningen af en fil, og for at bringe dens indhold på skærmen som billede. Hvis filen ikke indeholder nogen data, kan det give mærkelige effekter. Modstykket til *einlesen, hvormed man kan skrive billeder i en fil, er *speichern. Kun billeder, der er gemt på disketten med *speichern eller et lignende program, kan hentes igen med *einlesen.

*speichern:

Under gemning virker programmet lige modsat. Billedet hentes først med GET til en feltvariabel, så spørger programmet om et filnavn og åbner en fil med navnet til sekventiel udlæsning. Udlæsning af dataene til diskette sker

med en FOR-løkke, da filens længde jo er kendt (ud fra feltstørrelsen). Her anvendes igen to nye kommandoer.

PRINT #filnummer,USING n$;udtryk1,udtryk2,...
udlæser værdierne af udtrykkene til en sekventiel fil, n$ er en streng, der nøjagtigt angiver, hvordan tallene eller strengene skal udlæses. (Denne mulighed benyttes ikke i programeksemplet.)

Før et element udlæses af *bild&, skal det forvandles til en streng. Dertil anvendes konverteringsfunktionen MKL$. Følgende funktioner bruges til forvandling af numeriske tegn til tegnstrenge:

v$ = CHR$(n)	værdi af et tegns ASCII-kode til 1-bytes tegnstreng
v$ = MKI$(A%)	kort heltal til 2-bytes tegnstreng
v$ = MKL$(A&)	langt heltal til 4-bytes tegnstreng
v$ = MKS$(A!)	enkelt-præcisions tal til 4-bytes tegnstreng
v$ = MKD$(A#)	dobbelt-præcisons tal til 8-bytes tegnstreng

I programdelen *speichern går man altså den modsatte vej og forandrer tal til tegnstrenge, der kan gemmes. Også her slettes feltet *bild& fra hukommelsen med ERASE, fordi det er meget pladskrævende. Hvor meget plads viser sig i dimensioneringen af feltet (med DIM), der ikke sker i starten, men med en separat subrutine.

*bilddim:

Tallene til dimensioneringen blev beregnet efter formlen i kapitel 2.6. bild&-*billede& har 9510 elementer, og fordi hvert element er et langt heltal, kræves 38040 bytes.

For ikke at indlæse et billede, før noget vises på skærmen, males i programdelen *zeichnen et ret simpelt billede (tre højhuse med en række vinduer samt et lyn på venstre side). Billedet kan gemmes eller erstattes af et andet, ved at indlæse det andet billede.

*zeichen

Bemærk den palet, der anvendes i programmet. Farverne 18 og 19 har jo samme værdi! Hvis billedet kaldes på normal vis, kan man ikke se lynet, fordi det har samme farve som himlen. Lynets fremkomst er gemt i programafsnittet *lau, hvortil hoppes med regelmæssigt tidsinterval. Hvordan kan det lade sig gøre? Jo, man kunne f.eks skrive en nøjagtigt tilpasset tids-løkke. Det er dog besværligt og overflødigt, Basic har en tilsvarende kommando.

ON TIMER(n) GOSUB hopmærke eller linienummer
hvert n'te sekund forgrener programmet til hopmærke/linie

Man skal dog kalde endnu en kommando for at sætte interrupt-reaktionsevnen i kraft.

TIMER ON
ON TIMER aktiveres
TIMER OFF
virkningen af ON TIMER frakobles
TIMER STOP
virkningen af ON TIMER afbrydes forbigående

Ved TIMER STOP undersøger fortolkeren, om den, med ON TIMER fastsatte tid er udløbet. Så snart TIMER ON fremkommer i programforløbet, forgrenes til den tilsvarende rutine. Når man beordrer TIMER OFF, standses "stopuret" og går først i gang efter et TIMER ON. I slutningen af kapitlet er endnu et TIMER-eksempel, til eksperimenter.

I subrutinen *lau hoppes der, efter timerens udløb, til den anden palet. Herved ændres farve 18 til gul, farve 19 til blå, og lynet fremkommer. Derefter genoprettes den normale palet. Det er vigtigt, at timeren frakobles, når programmet befinder sig i en anden programdel. Hvis timeren er i gang under gemning af et billede, kan det lave et kønt rod i dataene. Billedet vil ikke være til at kende igen. Programlinien

IF tim THEN TIMER OFF: tim=0
frakobler timeren, hvilket noteres i variablen tim.

*farbtafel:

Selv om programmet benytter en SCREEN med 32-farve mode, anvendes kun 9 farver. Man kan altså lave billedet mere kulørt. Prøv bare.

Den viste måde at gemme og indlæse grafik på, er behagelig nem; men den slags filer kan kun benyttes af Basic-programmer og ikke af Graficraft eller Deluxe Paint. Hvordan man lagrer og indlæser grafik i et format, der passer til dem (det såkaldte IFF-format), vises i kapitel 3.3

Før vi bygger videre på det viste program, er her det lovede øvelsesprogram til TIMER-sætningen

REM mauer
'P.2.7-2 TIMER-Funktion

REM P.2.8

Men hvordan opdager programmet, hvor længe det har kørt? Naturligvis ved hjælp af en funktion.

> **v=TIMER**
> overgiver det antal sekunder, der er gået siden midnat eller siden computeren blev tændt

I programmet *Mauer hentes tidsværdien med TIMER-funktionen ved programmets start og lagres i variablen t. Ved hvert af timerens kald af subrutinen *zeit trækkes t fra den aktuelle tid, og resultatet udlæses på skærmen.

2.8 PAINTAmiga

-rutinen, er der alle steder sat "pointere", som sørger for, at der ved den anden hovedfordeler af *MausKontrolle, forgrenes til den rigtigte subrutine.

REM PAINTAmiga
REM P.2.8

Programmet har, i sin fuldstændige version, en længde på over 15 Kbytes og er stort og uoverskueligt. For at lette forståelsen, er de vigtigste funktioner vist i et forenklet skema:

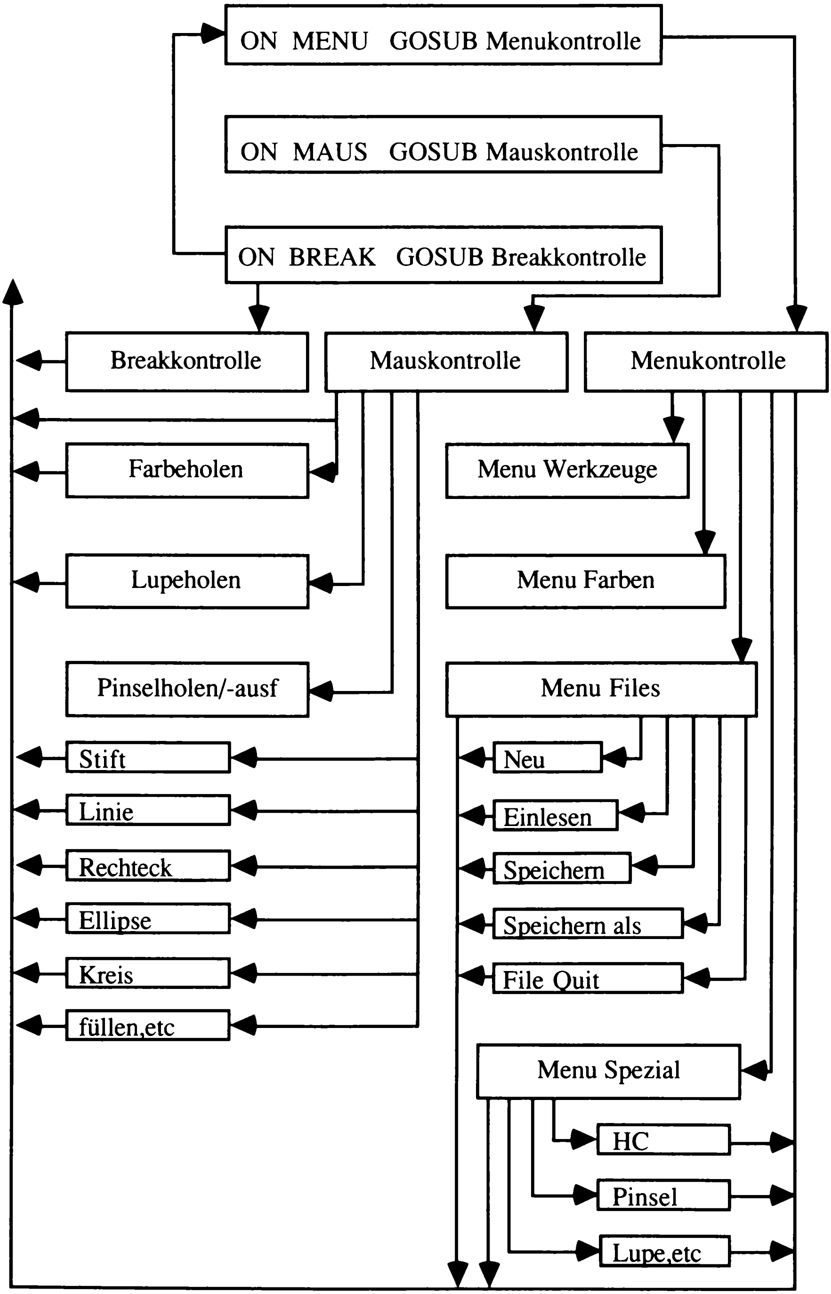


Fig 2.1 Udførelsesskema for PAINTAmiga

2.8.1 PAINTAmiga i detaljer

Et kort overblik over programmet, der venter (SLEEP) i en WHILE-WEND løkke, indtil et interrupt indtræffer. Basic forgrener på grund af ON-sætningen til rette rutine for denne event. De tre events, der kan fremkalde et interrupt, er højre museknap (ON MENU GOSUB), venstre musetast (ON MOUSE GOSUB)) og tasterne til programafbrydelse (ON BREAK GOSUB). De sidste bevirker, at programmet kører upåvirket videre, fordi der kun står RETURN i subrutinen. Programmet lader sig ikke afbryde.

Ved valg af et programpunkt, i en pull-down menu, med højre museknap, hopper programmet til subrutinen *MenuKontrolle. Der sker den videre forgrening til de subrutiner, som hører til de enkelte menu-kommandoer. Heri sker hovedsagelig forberedelserne til arbejdet med musen (venstre knap). Et blik på rutinerne viser, at med undtagelse af *Hardcopy-rutinen, er der alle steder sat "pointere", som sørger for, at der ved den anden hovedfordeler af *MausKontrolle, forgrenes til den rigtigte subrutine.

Nu detaljerne: i begyndelsen af programmet er der ikke noget specielt, men så kommer fire nye sætninger:

ON MENU GOSUB hopmærke
forgrener til hopmærke, når et menupunkt er valgt
ON BREAK GOSUB hopmærke
forgrener til hopmærke ved tryk på CTRL-C,Amiga- eller valg af Stop-kommandoen fra RUN-menuen

For at kommandoerne skal kunne aktiveres eller inaktiveres, kræves - som kendt fra ON MOUSE og ON TIMER -, andre kommandoer:

BREAK ON
aktiverer ON BREAK
BREAK OFF
inaktiverer ON BREAK
BREAK STOP
ophæver virkningen af ON BREAK midlertidigt
MENU ON
aktiverer ON MENU
MENU OFF
inaktiverer ON MENU
MENU STOP
ophæver virkningen af ON MENU forbigående

(MENU RESET-sætningent, der følger efter ON-kommandioen forklares sammen med subrutinen *MenuEin.)

Nu kommer vi til de enkelte subrutiner i programmet

***Mauskontrolle:** Henter musens startposition, der kræves til forskellige værktøjer, og forgrener så til de enkelte subrutiner.

***Lupeholen:** Hvis luppen er indkoblet, forgrenes der til *Lupenbild. Så afprøves, om der er klikket i lup-vindues området. Da det ville føre til et crash, ignoreres klikket i så fald. Den firkant, der skal forstørres, gemmes og forsynes med en ramme. Så åbnes vindue 3 og til sidst aktiveres vindue 2 igen.

Derefter bliver de enkelte billedpunkter, i det markerede område, læst og tegnet 16 gange forstørret i lup-vinduet; dvs. hvert punkt i originalen bliver til et 4 punkter bredt kvadrat i lupen (som forstørrelse almindeligvis defineres (lineært), svarer det kun til en 4 ganges forstørrelse, O.A.). Hertil anvendes følgende funktion:

v=POINT (x,y)
giver farvekendetegnet (ifølge PALETTE) for punktet på de angivne koordinater (x,y)

***Lupenbild:** Først undersøges, om musen blev klikket til højre for lupvinduet, hvis ja forgrenes til *Lupezu-, hvis ikke forvandles musepositionen, så det forstørrede punkt kommer til den rette position. Så afsættes det forstørrede og det normalt store punkt efter hinanden.

***Lupezu:** Det nytegnede billedudsnit gemmes, den tidligere tilstand genoprettes, og så sættes det nye billedudsnit. Herved forsvinder tillige rektanglet omkring det forstørrede udsnit. Så slettes standard-penslerne, da de blev misbrugt under billedets midlertidige gemning, og indlæses på ny.

***neufarbeyr:** Der åbnes et vindue, som fyldes med den aktuelle penselfarve, derefter udskrives teksten "RGB" med henholdsvis et plustegn (+) over og et minustegn (-) under.

***neufarbeholen:** Alt efter grafik-cursorens position i det øjeblik, der klikkedes på musen, forhøjes eller formindskes en af de tre farvekomponenter rød, grøn og blå med 0.067 (en sekstendedel). Positionen undersøges ved hjælp af en parentetisk IF-THEN-ELSE konstruktion. Den nye farveindstilling aktiveres med PALETTE.

***Farbeholent:** MOUSE(3) giver x-koordinater og MOUSE(4) y-koordinater til det sidste museklik. Hvis brugeren har klikket i paletten, bliver denne farve til den aktuelle penselfarve.

***pinselfarbe:** Vindue 3 åbnes, den aktuelle pensel vises i den sidste valgte farve. Når den ændrede pensel er gemt, lukkes vindue 3 igen.

***pinselholen:** Hvis musetasten aktiveres, medens musens pil er uden for pensevinduet, lukkes det igen. Klikkes i området y beregnes ud fra x-positionen, hvilken pensel, der er valgt. Vælger brugeren at indlæse eller konstruere en pensel, gemmes en tilsvarende værdi i variablen *pinsel.

***pinsellesen:** De tidligere gemte valgpensler (frit valgbare pensler) listes i vindue 3. Kommandoen FILES "WP" betyder, at indholdet af skuffen WP listes i det aktuelle vindue. Underprogrammet spørger til slut efter navnet på den pensel, der skal indlæses. Trykkes RETURN, uden at der er indtastet et navn, lukkes penselvinduet, og der hoppes tilbage til hovedprogrammet. Den ønskede pensel indlæses i subrutinen *wahlpins.

***pinselkonstr:** Er værktøjstypen stillet på 10 (intet værktøj), tegnes et rektangel, hvori den nye pensel kan oprettes. Så følger fire IF-THEN sætninger. Alt efter hvor museklikket sker, ignoreres indtastningen, penselen gemmes, eller den allerede tegnede pensel slettes. Ved gemning af en ny pensel gælder det samme for fil-navnet, som er sagt i det foregående afsnit om indlæsning. I den følgende WHILE-WEND løkke tegnes sluttelig den nye eller ændrede pensel.

***Stiftpinsel:** Når variablen p1 er sand, hentes den aktuelle pensel i variablen var og anbringes således på skærmen med PUT, at musens pil er midt i penslen. Mode for PUT er i så fald OR. Penslen knyttes altså til et allerede tilstedeværende billede. (Se også kapitel 1.6.) Prøv bare en anden mode, f.eks PSET, det resultat synes måske mere passende.

Programmet PAINTAmiga har en række værktøjer til maling. Til næsten alle er det nødvendigt at markere en linie eller et rektangel, medens musen bevæges med nedtrykket venstre knap. Markeringen overtages af rutinen *untermaus.

***untermaus:** Først fastslås den aktuelle museposition, og det øjeblikkelige skærmindhold gemmes ved hjælp af GET i variablen *Bild. Så tegnes en linie eller et rektangel i den aktuelle farve og slettes igen efter en kort pause, idet det gamle skærmbilledindhold genoprettes med PUT. Såsnart der ikke trykkes på museknappen, forlades WHILE-løkken.

***Kopie:** Der kræves to rutiner til at realisere dette værktøj. Den første gemmer indholdet af rektanglet fra *Untermaus. Da størrelsen af rektangelet er variabel, reserveres herved tilstrækkelig plads til hele skærmen. Pointeren ko sættes så til -1 (logisk sand), så programmet under det næste gennemløb skal gå til subrutinen *Kopie1.
***Kopie1:** Den anden kopieringsrutine aktiveres igen af et museklik. Billedet, der er gemt i *Kopie følger musens pil, sålænge museknappen holdes nedtrykket. Ved hvert gennemløb af WHILE-sløjfen kaldes PUT-kommandoen to gange. Første gang fremkommer billedet, og slettes efter en kort ventetid igen af den anden PUT-sætning. Når musetasten slippes, forlades WHILE-løkken og billedet placeres, en sidste gang med PUT, på sin nye plads.

***MenuEin:** Amiga-Basic klarer også anvendelsen af pull-down menuer i egne programmer. Dertil kræves følgende Basic-kommandoer:

> **MENU betegnelse,menupunkt,status,titel**
> frembringer et nyt menupunkt nummerets betegnelse, eller en ny menu med den anførte titel
> **MENU RESET**
> sletter alle af programmet definerede menuer fra listen og viser atter Amiga-Basics menuer

Hvis man betragter subrutinen *MenuEin, ser man tallene ét til fire anvendt som betegnelser. Der defineres altså fire menuer. (Basic kan maksimalt administrere ti menuer.) Antallet af menupunkter er forskelligt og ligger mellem ni og nul i vort program. (Basic tillader indtil 19 punkter.)

Angives nummeret nul som menupunkt, frembringes en ny menu, ellers får en allerede tilstedeværende menu et ny punkt. Status er altid lig med 1 i dette program, dvs. alle menuer og menupunkter er aktive.

***MenuFarben:** En WHILE-WEND løkke tegner 32 små rektangler i menubjælken, som indeholder de 32 mulige farver.

***zurueck:** Dette mikroskopiske underprogram kan have stor effekt. Som rutinens navn siger, henter den et eventuelt ødelagt billede tilbage. Bedre sagt bringes billedet tilbage i den tilstand, det var i før sidste ændring. Hertil anvendes atter GET og PUT. I rutinen *anfang gemmes hele skærmbilledindholdet som Bild1-*billede1 med en GET-sætning ved hvert programgennemløb. Hver gang musen aktiveres, kaldes subrutinen *Zurueck. Hvis menupunktet *zurueck ikke er blevet valgt, så slettes *Bild1 først, og så gemmes skærmbilledets nuværende tilstand igen som *Bild1 men hvis man ønsker at hente foregående billede tilbage, bringes det atter på skærmen fra *Bild1 med en PUT-kommando.

***pinsel:** Åbner vindue 3 og viser de fem mulige penselvalg i det nye vindue. Som videre valgmuligheder udlæses punkterne *holen (til indlæsning af en valgpensel) og *fertigen (til konstruktion af en valgpensel) i penselvinduet.

***fensterau:** Det område af billedet, som det nye vindue skal dukke op i, gemmes som *Bild2. Vindue 3 åbnes med en WINDOW-sætning

***fensterzu:** Vindue 3 lukkes med sætningen WINDOW CLOSE 3. Derefter genoprettes de bagved liggende dele af billedet med PUT-sætningen. (Denne del blev jo gemt i *fensterauf i variablen *Bild2.)

***Neuy:** Foreligger ingen tegning, sker et hop tibage til begyndelsen af hovedprogrammet. Ellers bliver skærmbilledet gemt som *Bild. Understreget af et akustisk signal spørges, om billedet virkelig skal slettes og et nyt påbegyndes. Programmet venter på en indtastning (S eller L). Der kan benyttes store og små bogstaver. Følgende funktion omdanner alle bogstaver til store, før de kontrolleres:

v$=UCASE$(x$)
forvandler tegnstrenge x$ til store bogstaver

***Pinselstandard:** Her indlæses de tre standardpensler og en valgpensel fra disketten. Da subrutinen er opdelt i tre, kan subrutinen *Wahlpins tilgås alene. Den mulighed benyttes ved ændring af sidste pensel.

***Farbtafel:** Her læses farvekomponenterne for paletfarverne fra DATA-linierne og tildeles de tre indicerede variabler rød!(), grøn!() og blå!(). Variablerne anvendes i de følgende PALETTE-sætninger til at definere farverne.

***Speichernals:** Rutinen gemmer det aktuelle billede på disketten. Den svarer til den fra det sidste kapitel. Vælger brugeren kun Save og ikke Save AS, spørges ikke om et filnavn. I stedet anvendes det filnavn, der siden sidste billedindlæsning har været gemt i variablen filename$

2.8.2 PAINTAmiga kører

Vil man benytte PAINTAmiga hyppigt, bør man gemme det på en særlig diskette. (Et billede optager næsten 40 kytes). Man bør tage hensyn til følgende punkter. AmigaBasic skal være på samme diskette som PAINTAmiga. Anlæg to tomme skuffer og kald dem (med Rename fra Workbench-Menu Project) SP (= standardpensel) og VP (= valgpensel).

En normal musepil er uegnet til præcist grafikarbejde. Man bør oprette en, som også kan bruges til andre ting. Indlæg Workbench-disketten og åbn ikonet med et dobbeltklik. Start programmet Preferences og vælg omskifteren Edit Pointer. Opret så følgende musepil:

```
000021000120000
000021000120000
000021000120000
000021000120000
222221000122222
111111000111111
000000000000000
000000030000000
000000000000000
111111000111111
222221000122222
000021000120000
000021000120000
000021000120000
000021000120000
```

0 = blå, 1 = brun, 2 = sort, 3 = rød

Fig.2.2: Skema til en forbedret musepil

Det røde kvadrat i midten gøres til det "varme punkt", ved at klikke på omskifteren Set Point og på punktet. (Man skal ikke lade sig forvirre af skemategningens langstrakte format. I virkeligheden bliver musepilen nøjagtig kvadratisk.) Aktiver nu omskifteren OK og (I Preferences- hovedvinduet) omskifteren Save.

Nu har vi den rette musepil og skufferne til programmet PAINTAmiga, men skufferne med penslerne er stadigvæk tomme. Hvis man starter programmet nu, søger det efter standardpenslerne i skuffen SP, men finder dem ikke og afbryder programmet med fejlmeldingen File not found. Standard-penslerne skal altså først indlæses. Det kræver et tegneprogram, men sådan et har vi jo allerede. Det skal blot ændres til dette specielle formål.

Følgende programlinier skal ændres for at konstruere standard-penslen.:

ved begyndelse GOSUB penselstandard i REM GOSUB penselstandard
ved pensel
PUT (14,0),pens1,PSET.. i REM PUT (14,0),pens1, PSET
PUT (44,0),............ i REM PUT (44,0)............

ved penselkonstr
IF posy&AND posy28.......:PL$ = "WP/":.......
i IF posy&AND posy28...............:PL$ = "SP/":.......

Indtast ændringerne og start PAINTAmiga. Vent til Titelbjælken PAINTAmiga af.... fremkommer. Vælg så menupunktet *Pinsel fra pull-down menuen *SPEZIAL. Penselvinduet åbner sig. I det nye vindue klikkes på "fertigen" med musen og klikker en gang til. Nu fremkommer et næsten kvadratisk tegnefelt i vinduet. Heri tegnes en næsten udfyldt cirkel. Synes man ikke om tegningen, kan den slettes med et klik på ordet "*loeschen" og begynde forfra. Når man er færdig, klikkes på OK-feltet. Tegningerne forsvinder, og man bliver opfordret til at indtaste et navn. Skriv *pins1 og tyk på RETURN-tasten. Så er den første pensel færdig. Man gentager processen, tegner en stor udfyldt cirkel,og kalder den *pins2. Tredje gang tegnes et kors og gemmer det som *pins3. Så en gang til med et spørgsmålstegn og som filnavn *pins?.

Afslut nu PAINTAmiga med Quit fra Project-Menu. Så er vi igen i Amiga-Basic. Slet programmet fra hukommelsen med New. Den uændrede version har vi på disketten. Og nu er vi klar til at anvende PAINTAmiga.

Start programmet. Før det kommer frem, skal bemærkes, at man altid bør gå roligt frem for at opnå det ønskede resultat. Tænk på, at vi arbejder med et Basic-program og ikke med et optimeret maskinsprogsprogram. PAINTAmiga er langsomt! Derfor skal man lade fingeren hvile på museknappen, indtil der sker noget - også selv om det varer et halvt sekund eller længere. Og når man slipper museknappen, skal man lade musen stå et øjeblik på den gamle position. Tager man hensyn til det, vil man let kunne opnå pæne billeder med PAINTAmiga.

2.8.3 PAINTAmiga-håndbog

Her er en lille håndbog, som forklarer de vigtigste kommandoer og virkningerne.

Den første menu, Project, bør være kendt fra andre programmer. Jeg går derfor ikke videre ind på dette punkt. Værktøjs-menuen er mere interessant.

Stift: Efter valget af menupunktet stift, kan man tegne på fri hånd. Dvs. man kan anbringe vilkårlige farvepletter og kurver ved at trykke på museknappen og bevæge musen. Det er i øvrigt også den eneste mode i programmet, som penslerne arbejder i. Alle andre værktøjer benytter den lille, punktformede pensel.

Linie: Værktøjet benyttes til at tegne lige linier. Tryk på det punkt, hvor linien skal begynde, og kør langsomt med nedtrykket museknap til det punkt, hvor linien skal holde op. Slip knappen, og straks tegner programmet en linie.

***Rechteck:** Klik i et hjørne af det ønskede rektangel og bevæg musens pil over i modstående hjørne, medens knappen holdes nede. Når man slipper den, fremkommer det nye rektangel. Senere kan man udfylde det med menupunktet *fuellen; men man må ikke skifte farve, ellers løber farven ud over hele billedet. For at udfylde, skal man bare klikke i rektanglet. Udfyldning virker på samme måde med cirkler og ellipser.

Ellipse: kommandoen fungerer ligesom rektangel. Når man slipper museknappen, tegnes blot en ellipse inden for den rektangulære begrænsning.

***Kreis:** Tryk på museknappen i centrum af den ønskede cirkel, bevæg så musen og slip museknappen, når cirklen når den ønskede radius.

***Radierer:** Med kommandoen går man frem, som om man ville tegne et rektangel. Så snart man slipper venstre museknap, slettes alt (overmales med hvidt) inden for det rektangulære område.

***Flaeche:** Det er svært at beskrive dette værktøj. Prøv det. Klik på et vilkårligt sted og bevæg musen med nedtrykket knap.

***Kopie:** Her skal markeres et rektangel, slip museknappen og vent et øjeblik. Tryk på tasten og hold den nede. En kopi hænger fast på musepilen og følger alle bevægelser. Når tasten slippes, bliver kopien liggende i billedet.

Menuen *Farben vælger penselsfarve. Den har et tomt område (uden navn). Når den navnløse kommando vælges, kommer alle 32 farver på menulinien. Man behøver blot at klikke på den korrekte farve med musens pil.
Vi kommer nu til de mest interessante, specialmenuer

***zurueck:** Synes man ikke om sidste ændring af billedet, vælges *zurueck og klikker med musen et tilfældigt sted, og den foretagne ændring er væk. Funktionen kan ikke vælges to gange efter hinanden.

***Pinsel:** Vælges dette punkt, åbnes et nyt vindue i skærmens øverste venstre hjørne. I penselvinduet har man flere valgmuligheder. Et klik på en af de fem pensler, giver en ny pensel. Vinduet lukkes så igen. Klikker man på "*Text holen", slettes penselvinduets skærm. Efter endnu et klik listes alle gemte valgpensler i penselvinduet. Så indtastes navnet på den ønskede pensel; som indlæses som femte pensel. Hvis man fortryder, trykkes RETURN, og penselvinduet lukkes igen.

Den tredje mulighed er "*fertigen". Her fremkommer efter andet museklik et rektangel, hvori man kan konstruere en ny pensel. Er man er tilfreds, klikker man på OK, og bliver bedt om et navn, som den nye pensel skal gemmes under. Man kan fortryde og blot trykke RETURN, så lagres penslen ikke. Vil man lukke penselvinduet uden at gemme en ny pensel, kan man klikke uden for vinduet, så det lukker sig igen.

***Lupe ob:** Når man vælger denne funktion, sker der ingenting; men trykker man venstre musetast i billedet, indrammes området omkring musens pil af et rektangel, og et vindue åbner sig. Heri vises det indrammede områdes indhold i 16 gange forstørrelse. Hvis man nu klikker inden for luppen, fremkommer en lille firkant, og et punkt af den aktuelle farve sættes i originalbilledet. Når man er færdig, klikker man til højre for lup-vinduet i billedet, og luppen forsvinder.

***Lupe un:** Denne funktion svarer til *Lupe ob, men lup-vinduet åbnes i nederste halvdel af billedet. Man skal lægge mærke til, at området under lup-vinduet kun må markeres til forstørrelsen.

SW-HC: Denne kommando er beregnet på grafik-undtryk, der endnu ikke er implementeret i denne version af PAINTAmiga.

***NeuFarb:** Med kommandoen kan man ændre den aktuelle penselfarve. - Vælger man det menupunkt, åbnes et vindue, som indeholder to bjælker med den aktuelle penselfarve. Man kan øge rød-, grøn- eller blå-andelen af farven ved at klikke på plus over R, G eller B. Et klik på minus under samme bogstaver mindsker den tilsvarende andel. Således kan man se Amigaens 4096 farver.

Held og lykke med billederne.

3 FILER

Bemærk hvordan man behandler filer på computeren. Det er ikke meningen at opbygge en komplet database, vi vi blot se på grundtrækkene; bl.a. hvordan man kan gemme billeder fra Basic på diskette i et format, der også kan forstås af andre programmer som Deluxe Paint og Grafikcraft.

3.1 Printning af tekst

Det forbavser måske at se ordet "printning" i et afsnit om filer; men sagen er, at Amigaen behandler en printer, som om den var en fil, man ganske vist kan skrive noget i, men aldrig kan læse fra.

Før man kan skrive i en fil, skal den åbnes. Det samme gælder printeren. Nogle kommandoer sender dog kun output til en printer og aldrig til skærmen eller en fil. Så er det overflødigt at åbne "printerfilen" først.

Når man ikke kan finde en fejl i et program, er det ofte lettere at lede efter den i en udprintet listning. Det printes med en særlige printerkommando.

> **LLIST område-område**
> lister et Basic-program på printeren

LLIST opfører sig nøjagtigt ligesom kommandoen LIST (der blev beskrevet i Kapitel 1), blot sendes listningen til printeren i stedet. LLIST printer hele programmet, LLIST-500 printer linierne indtil nr. 500 osv.

Med PRINT sender man alt output til skærmen. Der er også en anden kommando, der istedet sender det til en printer.

> **LPRINT liste af udtryk**
> **LPRINT USING v$; liste af udtryk**
> udlæser værdierne af liste af udtryk på printeren.

En printer har som skærmen en begrænset bredde. Hvis den liste, der udprintes, har linier på mere end 80 tegn, gås automatisk til en ny linie. 80-tegns bredden er blot en forudindstilling, man kan lave om på.

WIDTH LPRINT bredde, skriveområde
WIDTH #filnummer,bredde,skriveområde
WIDTH apparat,bredde,skriveområde
fastlægger, efter hvilken linielængde, der automatisk skal skiftes linie på en printer, i en fil eller med et andet apparat

Den sidste angivelse, skrivebredde, i en WIDTH-sætning fastlægger skriveområdets linielængde. Skriveområder forholder sig som tabulatorer på en skrivemaskine. Hvis man adskiller elementerne bag PRINT eller LPRINT med kommaer, hopper Basic til slutningen af det pågældende skriveområde, før det udlæser næste værdi.

Man kan prøve virkningerne af WIDTH-sætningerne på skærmen. Indtast WIDTH 20 og derefter 1234567890123456789012345 i direkte funktionsmåde. Udlæsningen af teksten afbrydes allerede efter det 20. ciffer.

Følgende programeksempel viser en mem anvendelse af LPRINT-sætningen. Det lister hele det tegnsæt, printeren har indbygget.

REM Zeichen
REM p.3.1-1

Programmet udprinter tegnene med ASCII-koderne 33 til 127 og 161-255. Liniebredden er indstillet til 20 tegn, ikke med en WIDTH-sætning; men med en IF-kommando, der afprøver, om det 20. sted på linien allerede er nået. I så fald udlæses en linieskift-kode CHR$(13).

v=LPOS(n)
angiver positionen af det sidst trykte tegn

Parameteren n er en værdi uden betydning, der alligevel ikke må udelades.

Nu hvor vi ved, hvordan man sender tekster til printeren, kommer den interessante del, grafik-udprintning

3.2 Hardcopy

Det afhænger selvfølgelig af printeren, hvad man kan byde den. En matrixprinter er nødvendig til grafik-printning, en typehjulsprinter kan kun bruges til tekst-udlæsning.

De anførte programmer og sætninger er testet med EPSON EX-800 printeren. Det er en 9-nåls matrixprinter, der med ekstraudstyr kan printe i farver.

(Så længe man kun printer i én farve er den kompatibel med EPSON FX-80.) Man skal ikke give op, hvis man har en anden printer. Moderne matrixprintere ligner hinanden så meget, at der kun er programtekniske detaljer til forskel. Når man forstår strukturen i følgende programmer, kan man for det meste tilpasse dem til andre printere. Det kræver en printer-manual og en smule tid.

Ved grafikudprintning er der to vigtige nyheder for en Basic-programmør: styringskoderne, der omstiller printeren til grafik-udprintning og tilgangen til de enkelte printernåle. Styringskoderne kommenteres i de følgende programeksempler, hvorved det er let at tilpasse dem til en anden printer.
Se først på nålestyringen. I printerens hoved ligger de nåle eller stifter, der sørger for printningen, lodret over hinanden. Grafik-udprintning sker som med tekst linie for linie. I grafik-mode anvendes de øverste otte nåle af printerhovedet. Det betyder, at der for hver spalte af en sådan grafiklinie, skal angives til printeren, hvilke af de otte nåle, der skal bearbejde farvebåndet. Det sker i form af et tal mellem 0 og 255. Tallet 255, som man møder hele tiden under programmering, betyder 8 bits eller en byte.

For at konstatere det tal, der skal sendes til printeren, tildeles hver af de øverste otte nåle et tal, således:

```
o 2^7 = 128
o 2^6 =  64
o 2^5 =  32
o 2^4 =  16
o 2^3 =   8
o 2^2 =   4
o 2^1 =   2
o 2^0 =   1
```

For at få printet en bestemt kombination af nåle, skal man addere tallene for de nåle, der skal printe. Hvis der ikke er nogen nåle i en række, der skal printe, angives værdien 0, og hvis alle nåle skal printe, tæller man alle værdierne sammen og får 255. Her er et par eksempler, der anskueliggør det hele:

+128	0	+128	0
0	+64	+64	0
0	0	0	+32
+16	0	0	+16
0	+8	0	+8
0	0	+4	0
+2	0	0	+2
0	+1	+1	0
Sum: 146	Sum: 73	Sum: 197	Sum: 58

Nu et eksempel. (NB: dette eksempel virker kun, hvis den anvendte printer er forbundet med Amigaens serielle port!)

REM Amiga-K
REM P.3.2-1

I første linie åbnes en fil med nummeret 2. Filen får navnet "PAR:", det betyder, at alle data, vi sender til filen, går til det parallele interface og dermed til den tilsluttede printer. Ifølge Basic-manualen skulle man egentlig anvende "LPT1:". Det er O.K. til tekstudlæsning; men til grafik bruges "PAR:".

I den anden linie sendes en styrekode med PRINT#2, der ændrer linieafstanden, så grafiklinierne printes tæt og ikke med en hvid stribe imellem.

Den følgende FOR-løkke sender grafiklinien til printeren ti gange efter hinanden; men først sendes en række styrekoder til printeren, der fortæller, at der følger 17 bytes (17 spalter) med grafikdata efter hver af dem.
I den indre FOR-løkke aflæses de 17 tal, der sendes til printeren og styrer, hvilke nåle, der skal trykkes i hver spalte, med kommandoen READ fra DATA-kommandoerne i slutningen af programmet.

> **READ variabel,variabel,...**
> læser data fra DATA-linier og tildeler dem til variabler

De værdier, READ læser, skal lægges parat med DATA-kommandoerne.

> **DATA konstant,konstant,...**
> lægger konstanter ind i hukommelsen, som programmet kan læse med READ

Her er et lille eksempel:

REM farbe
REM P.3.2-2

Programmet indlæser dataene til farvebestanddelene af fire af palettens farver fra DATA-sætningerne og anvender dem i PALETTE-sætningen. Bemærk rækkefølgen af numeriske værdier og strenge, der skal stemme overens i DATA'ene og i READ-sætningen.

Tilbage til printerprogrammet. I den indre sløjfe indlæses DATA-værdierne og de udprintes spaltevis. Hver gang en linie er afsluttet, kaldes kommandoen RESTORE.

RESTORE mærke
læsepointer for READ bliver genindstillet

READ begynder med at læse data ved den første DATA-sætning, og tager dem i rækkefølge. Med RESTORE sørges for, at den næste READ-sætning begynder med første DATA-sætning eller med den DATA-sætning, der står bag mærke.

Den sidste printerkommando i programmet sætter printeren i "normaltilstand" og tømmer printerbufferen. Så lukkes filen, og programmet er færdigt.

Om alt går vel, vil dette program printe Amiga-tegnet ti gange. Man kan roligt give sig til at konstruere et par andre billeder. Tegningerne kan også være flere spalter høje. Programmet skal så blot udvides med en løkke til.

Nu til grafik-Hardcopy programmet. På grund af Basic-grafikudprintningens lave hastighed, er programmet holdt så enkelt som muligt. Der printes uden mellemtoner, hvert punkt er enten sort eller hvidt. Alligevel kræver programmet ti minutter til den simpleste løsning, hvis der skal udprintes en grafik, der er 310 pixels bred og 180 pixels høj. Ikke ligefrem imponerende. Trods alt er programmet praktisk, fordi det er skrevet så universelt, at man kan inkorporere det uden større ændringer i mange forskellige Basic-programmer, der skal udprinte grafik.

Det første spørgsmål, der melder sig ved udprintningen af et vindue, er vinduesstørrelsen. Den kan fastslås med en speciel funktion.

v=WINDOW(n)
leverer, alt efter værdien af n (0-8), forskellig information om det aktuelle vindue

Format	Overgivne information	Eksempel
WINDOW(0)	Betegnelse for det valgte output-vindue, som ikke viser titlen med svag skrift Vindue 1 er Amigaens output-vindue	Direkte indlæsning: PRINT WINDOW(0) Resultat : 1
WINDOW(1)	Betegnelse for det aktuelle ouput-vindue, hvori print- og grafiske kommandoer virker	Direkte indlæsning: PRINT WINDOW(1) Resultat: 1 se 1. eksempel

WINDOW(2)	Bredde af det aktuelle output-vindue i pixels	Program: WINDOW 2,,,1 While 1 Locate 1,1
WINDOW(3)	Højde af det aktuelle out put-vindue i pixels	Print "Bredde= "window(2); Print "Højde= "WINDOW(3) WEND Man får resultatet, når man ændrer vinduesstørrelse med muse
WINDOW(4)	x-koordinater i det aktuelle output-vindue for udlæsningen af det næste tegn, i pixels	Program: FOR i = 1 TO 5 Print i;WINDOW(4);WINDOW(5); NEXT Resultat: 1 24 6 2 104 6 3 192 6...
WINDOW(5)	y koordinater i det aktuelle output-vindue for udlæsningen af det næste tegn, i pixels	eller, hvis det sidste mangler: 1 24 6 2 24 14 3 24 22
WINDOW(6)	Maksimale farveantal for det aktuelle ouput-vindue	Direkte indlæsning: Print WINDOW(6) Resultat:3 (uden baggrundsfarve)
WINDOW(7)	En adressepointer på INTUITION WINDOW-datasætningen for det aktuelle ouput-vindue	Program: eksempel WINDOW 2,,,8 WHILE 1 IF WINDOW(7)=0 THEN BEEP:EN Hvis vinduet lukkes, viser WINDOW(7) værdien 0
WINDOW(8)	Adressepointer på RASTPORT-datasætningen for det aktuelle output-vindue	Der er et eksempel i næste kapitel

Fig. 3.1: Forklaringer til Window-kommandoerne

REM SW-hardcopy
REM P.3.2-4

Programmet kort og enkelt. I begyndelsen fastlægges eller beregnes nogle tal, der er nødvendige for udtrykket. Så åbnes printerfilen, subrutinen *nadel

kaldes og printerfilen lukkes. Det meste af arbejdet udføres af *nadel. Den består af tre store FOR-løkker inden i hinanden. Den yderste gennemløbes én gang for hver printlinie, den næste én gang for hver printspalte, og den inderste én gang for hvert punkt i spalten. Alt efter om bredden af den pågældende grafik er større end 480 punkter, omstilles printeren i begyndelsen af den ydre løkke til en mode, hvori den kan udprinte enten maksimalt 480 eller 960 punkter i papirets fulde bredde.
Den indre FOR-løkke er den mest interessante. Efter hvert gennemløb af løkken (en gang for hver spalte i printlinien), sendes otte punkter, der ligger over hinanden, som ét tal (én ASCII-kode) til printeren. Den indre løkke bestemmer, hvilket tal, det skal være. Hertil sættes tallet (*wert) først til nul, derefter fastslås med POINT-funktionen, for hvert af de otte punkter i spalten, om det tilsvarende punkt på skærmen er. Hvis punktet er sat, adderes et tal til variablen *wert, der svarer til højden af punktet i printspalten. Der adderes 128 til det øverste punkt, 64 til det syvende osv. indtil det nederste punkt på printlinien, som er nummer 1.

Den indre FOR-løkke sluttes, og de således beregnede tal sendes som et tegn til printeren. Det sker én gang for hver spalte i en grafiklinie, og den mellemste FOR-løkkke er gennemløbet én gang. Ved slutningen af den sendes tillige koden CHR$(10) til printeren, den sørger for linieskiftet. Den yderste FOR-løkke fortsætter i næste printlinie (otte pixels længere nede).

Ved indtastningen må man ikke glemme semikolonet til sidst, ellers udprintes hver grafik-spalte som en ny linie.

Det er bedre at benytte et professionelt program, hvis man vil udprinte farvebilleder, egne programmer tager frygtelig lang tid. Næste afsnit viser, hvordan tegneprogrammet GrafiCraft (og andre programmer) printer billeder, frembragt med Basic.

3.3 IFF, chunks og den slags

IFF er en forkortelse for INTERCHANGEABLE FILE FORMAT og betyder "udskifteligt fil-format". IFF er en standard for grafik og musik på Amigaen. Software-firmaet Electronics Art i U.S.A. har udviklet en metode til at lagre data således, at de kan bearbejdes af alle programmer. Næsten alle programmer på Amigaen arbejder i dette format, når de gemmer og indlæser grafik.

Denne standard for dataudveksling definerer fil-typerne alt efter arten af data, der skal gemmes. Med kodeord i begyndelsen af den pågældende fil, vises hvilken type den tilhører.

Koden FTXT står for den formatterede tekst i en tekstfil, 8SVX for en 8-bits samplet Voice (stemme) i en fil for digitaliseret lyd, SMUS for Simple Musical Score i en musikfil og ILBM for Interleaved Bit Map (en grafik-fil), og det er ikke det hele. I dette kapitel vil vi se på filer af typen ILBM; vi omtaler kun den del af IFF-standarderne, der er vigtigtst for Basic-programmering.

3.3.1 Håndtering af hukommelsesadresser.

Ved IFF-filer er det vigtigt at håndtere hukommelsesadresser rigtigt med de kendte kommandoer PEEK og POKE. I modsætning til andre computere, kan man klare sig næsten uden PEEK og POKE på Amigaen, men ikke i disse specielle tilfælde.

BEMÆRK: netop kommandoen POKE er farlig, idet den kan ændre en vilkårlig adresse i Amigaens hukommelse, ligegyldigt hvad der stod i den. Ligger der en del af Basic-fortolkeren, kan effekten være katastrofal. Alle programmer, der indeholder POKE, skal indtastes og afprøves omhyggeligt, før de startes.

Men nu til definitionen af PEEK og POKE:

> **v=PEEK(adresse)**
> giver indholdet af den pågældende hukommelsesadresse som heltal mellem 0 og 255 (en byte)
> **POKE adresse,værdi**
> ændrer indholdet af hukommelsesadressen, idet den gemmer tallet værdi som en 8-bits binærværdi på adressen

Følgende eksempel demonstrerer begge de nye kommandoer:

REM reinraus
'P.3.3-1

Programmet aflæser hukommelseadresserne 10000 til 10015 og viser indholdet på skærmen. Med linien POKEi,PEEK(i) læses en værdi på en adresse og skrives atter ind i den. Linien har ingen anden effekt, end at demonstrere POKE-kommandoen.

PEEK og POKE har dog en stor mangel. De kan kun læse og skrive tal mellem 0 og 255 (en byte). Men i det følgende får vi ofte brug for større tal. Derfor er de to kommandoer, der arbejder med et helt tal, vigtigere.

v = PEEKW(adresse)
giver indholdet af den pågældende hukommelsesadresse som kort heltal (2 bytes)

Adresse skal her altid være et lige tal.

POKEW adresse,værdi
skriver værdi i hukommelsen som en 16-bits binærværdi på adressen

POKEW er modstykket til PEEKW. Adressen skal være et heltal.

Amiga-Basic kender endnu flere versioner af PEEK og POKE:

v = PEEKL(adresse)
giver indholdet af hukommelsen på den pågældende adresse som langt heltal (4 bytes)
POKEL adresse,værdi
skriver værdi i hukommelsen som en 32-bits binærværdi (4 bytes) på adressen.

Sætningen POKEL er modstykket til funktionen PEEKL. Adressen skal være et lige tal ved begge kommandoer

Følgende program læser direkte indholdet af forskellige typer variabler med PEEK-kommandoer:

REM VarSpeicher
'P.3.3-2

Først defineres de to variabler n% og n&, som lægges på en bestemt hukommelsesadresse af fortolkeren. Programmet fastslår hvor, med VARPTR-funktionen.

v = VARPTR(variabel)
giver de adresser, hvor værdien af de nævnte variabler skal gemmes

I de sidste to eksempler blev de nye kommandoer kun brugt til at vise indholdet af hukommelsesadresser; men vi kan også manipulere med adresserne. Følgende eksempel viser tillige, hvordan VARPTR anvendes på et felts elementer.

REM Feldspeicher
'P.3.3-3

Programmet tegner en cirkel i øverste venstre hjørne af skærmen. Skærmområdet hentes til feltvariablen *Bild% med GET. I den første FOR-løkke undersøges adresserne på de første 300 feltelementer med VARPTR, og de slettes med POKEW (sættes lig nul). Herigennem ændres farven af de tilsvarende punkter. Når *Bild% derefter atter er bragt på skærmen med PUT, er en del af cirklen sort. Processen gentages med lange heltal og feltvariablen *Bild&. Denne gang overskrives bits'ene i det øverste bitmap delvis. Resultatet er en cirkel, der er blevet rød, sort og blå. Hvis emnet interesserer, er der flere oplysninger i Kapitel 5.5

I det sidste eksempel med kommandoerne til hukommelsesmanipulation behandles strenge:

REM StringSpeicher
'P.3.3-4

Først tildeles en strengvariabel a$ ordet BMIGA, og indholdet vises på skærmen. En ny funktion finder så strengens adresse og det første tegn i den ændres.

> **v=SADD(tegnstrengsudtryk)**
> giver adressen, hvor det første tegn i en streng er gemt.

På dette sted skrives ASCII-koden for bogstavet A. Nu udlæses variablen a$ igen, og den er faktisk blevet til Amiga.
Nok om hukommelsesadresser; vi ser på det egentlige tema.

3.3.2 Indlæsning af IFF-grafik

denne information kan programmet forgrene til en passende rutine og behandle dataene der. Den følgende tabel viser et skema af en IFF-fil.

> Grundelement4 bytesstreng "FORM"
> 4 bytesnumeriskFORM-længde
> 4 bytesstrengFORM-betegnelse
>
> 1. chunk4 bytesstrengchunk-navn
> 4 bytesnumeriskchunk-længde "X"
> X byteschunk-indhold
> (eventuelt en byte som fyld)

2. chunk4 bytesstrengchunk-navn
4 bytesnumeriskchunk-længde "X"
X byteschunk-indhold
(eventuelt en byte som fyld)
3. chunk osv.

Tabel 3.2 Opbygningen af en IFF-fil.

Fyld-byten bag chunk-indholdet er påkrævet, når indholdet består af et ulige antal bytes. Den sørger for, at hver ny chunk begynder ved en lige adresse.

Måske siger al denne teori ikke læseren så meget; men vi vil belyse den med et eksempel. Billedet af Mona Lisa på tegneprogrammet Graficrafts diskette er sikkert bekendt. Vil laver et program til at se på chunk'ene i dette billede.

Først vil jeg sammenfatte den nødvendige viden om de funktioner, der kræves for at kunne indlæse variabler af forskellig typer, resp. skrive dem i en fil:

Udtryk > n	gemme	læse
Tal < 256	PRINT#1,CHR$(n);	n = ASC(INPUT$(1,1))
kortheltal	PRINT#1,MKI$(n%);	n = CVI(INPUT$(2,1))
langtheltal	PRINT#1,MKL$(n&);	n = CVL(INPUT$(4,1))
enkelt præcision	PRINT#1,MKS$(n!);	n = CVS(INPUT$(4,1))
dobbelt præcisions	PRINT#1,MKD$(a#);	n = CVD(INPUT$(8,1))
	n$ = "abc"PRINT#1,nn = INPUT$(3,1)	

Tabel 3.3: Nødvendige konverteringer ved ind- og udlæsning.

Indtast følgende program og gem det under titlen *IFFlesen. Hvis der ikke er tilsluttet nogen printer, så erstat hvert LPRINT med et PRINT.

REM IFFlesen
REM P.3.3-1

I programmet indlæses dataene i den rækkefølge, der blev vist ovenfor i skemaet over IFF-filer. Først åbnes en fil med det indtastede navn. Så indlæses de første fire bogstaver. Hvis de ikke lyder "FORM", er det ikke nogen IFF-fil, og programmet afsluttes. Efter indlæsningen af grundelementet, indlæser en WHILE-løkke de enkelte chunks. Efter at chunk-længden er fastlagt, forgrener programmet til rutinen *byteslesen. I denne programdel forhøjer vi længden med en fyld-byte, hvis den er ulige. Hertil anvendes bl.a. funktionen FIX.

v = FIX(x)
giver et tal, der fremkommer ved afskæring af cifrer efter kommaet (uden afrunding)

Når længden ligger fast, indlæses chunk-dataene i en FOR-løkke, og de første 100 bytes udlæses som tal. Så indlæses de resterende i to "bidder" og ignoreres. Til udlæsning af en byte som tal anvendes funktionen ASC.

v = ASC(x$)
giver ASCII-koden (mellem 0 og 255) for første tegn (byte) af x$

Før programmet startes, skal directoryet ændres. Så er det lettere at undersøge andre billeder som IFF-filer; men man skal stadig være forsigtig. Indtast i direkte funktionsmåde:

CHDIR "GrafiCraft"

Har man ombenævnt GrafiCraft-disketten, skal man bruge det nye navn og ikke "GrafiCraft".

Tilslut printeren og start programmet *IFFlesen. Skriv Mona Lisa, og vent på hvad der sker. Hvis der ikke er fejl i programmet, får man følgende:

Grundelement:	FORM
Formlaengde:	40156
Form-betegnelse:	ILBM
Chunk-navn:	BHMD
Chunk-laengde:	20
Chunk-indhold:	1 64 0 200 0 0 0 0 5 0 0 0 0 0 10 11 1 64
**	0 200
Chunk-navn:	CMAP
Chunk-laengde:	96
Chunk-indhold:	240 240 240 240 160 64 64 48 48 80 48...
	...til 112 48 128 96 64 80 80 48 48 48 48 0 0 0
Chunk-navn:	BODY
Chunk-laengde:	40000
Chunk-indhold:	0 16 0 0 128 0 0 0 0 10 0 0 0 0 0 0 0
Chunk-navn:	CAMG
Chunk-laengde:	4
Chunk-indhold:	0 0 64 0

Vi kan nu fastslå strukturen af en IFF-fil og behøver blot at finde ud af, hvilke funktioner der gemmes i hvilken fil. Andre billeder er stort set bygget ligesådan. I billederne Skier og NightFlight er en chunk til med navnet CCRT. Den er ansvarlig for farverotationen i billederne og interesserer os ikke her.

I en IFF-fils grundelement FORM står længden af filen på 40156 bytes, som formlæmgde samt formbetegnelsen ILBM, for "InterLeaved BitMap". Heraf ses, at det drejer sig om grafik.

Den første chunk er BitMap Headeren BMHD på 20 bytes. Da disse bytes skal udlæses enkeltvis af programmet, skal mange af dem atter sammensættes til ord (tal der består af to bytes).

ord	2 bytes	320	billedbredde
ord	2 bytes	200	billedhøjde
ord	2 bytes	0X	-position på skærmen
ord	2 bytes	0Y	-position på skærmen
byte		5	Antal bitplaner (til 32 farver)
byte		0	anvendte maske = ikke optaget
byte		0	hentede grafik = ikke optaget
byte		0	til fremtidige udv. = ikke optaget
ord	2 bytes	0	til maske = ikke optaget
byte		10	pixel-bredde forhold = 10:11
byte		11	pixel-højde med 320x200 skærm
ord	2 bytes	320	skærmbilled-bredde
ord	2 bytes	200	skærmbilled-højde

Tabel 3.4: Opbygningen af en BMHD-chunk

Foranstående liste gengiver, hvordan chunken ser ud, når de felter beregnes, der består af mere end én byte.

Nu kan vi begynde at udvikle et program til indlæsning af IFF-grafik. Med de fundne værdier for BMHD-chunken kan man opstille en rutine, som programmet forgrener til, når det møder en BMHD-chunk. Rutinen indlæser BMHD-chunken og analyserer den. Den kan findes i følgende listning fra label'en *bitmapdaten.

REM Teil1
REM

Begyndelsen af denne programdel svarer til programmet *IFFlesen. Overflødige chunks, dvs. chunks, der ikke kan viderebearbejdes, indlæses i streng-

variablerne rest$ og ignoreres. Så snart der er fundet en BMHD-chunk, kalder programmet subrutinen *bitmapdaten. Den indlæser værdierne af billedbredde, billedhøjde osv. i den ovenfor beskrevne rækkefølge, såfremt de er af interesse og læser forbi den (uinteressante) rest. Den SCREEN-mode, billedet senere skal vises i, giver sig selv ud fra skærmbredde og skærmhøjde.

Programmet hopper så til rutinen *fensterholen. Her fastslås bredde og højde af det aktuelle ouput-vindue med WINDOW(2) og WINDOW(3) og antallet af tilladte farver med WINDOW(6). Da værdien ikke tager hensyn til baggrundsfarven, forhøjes variablen *farben med én. Ud fra denne variabel beregnes med en FOR-løkke antallet af bitplaner, der er nødvendige for fremvisningen, og de gemmes i variablen *ebenen1. Med denne værdi afprøves i hovedprogrammet, om antallet af nødvendige bitplaner stemmer med antallet af i det aktuelle vindue. Hvis det er tilfældet, hopper programmet tilbage til *chunklesen. Har det aktuelle vindue et andet plan-antal, åbnes en ny skærm med et passende antal bitplaner og et vindue deri.

Den anden chunk hedder CMAP, for ColorMAP, og beskriver de farver, som paletten skal indeholde. Hver farve beskrives af tre på hinanden følgende bytes, der svarer til henholdvis rød- grøn- og blå-andelen af den tilsvarende farve. Chunken er for Mona Lisa-billedet 96 bytes lang. 96 bytes delt med tre bytes pr. farve giver 32 farver. Dette resultat kunne forventes, da der i BMHD-chunken var angivet fem bitplaner for billedet (altså 32 farver). Nu kan vi programmere anden del af læse-rutinen, der hedder *farbwert.

REM Teil 2

De 32 farvekomponenter indlæses i en FOR-løkke. Hver af dem deles med 240, før den gemmes i det tilsvarende felt. Det er nødvendigt, fordi der er én byte til rådighed for hver farveværdi i CMAP-chunken, og kun de højeste 4 bits heraf kan benyttes. Amigaen behøver jo kun fire bits pr. farveværdi. Den højest mulige farveværdi er 240 (når de højeste 4 bits er sat. Divisionen med 240 forvandler det heltal, der er gemt i CMAP-chunken til den tilsvarende brøk mellem 0 og 1, som PALETTE-sætningen forventer. De 16 mulige værdier af bytes'ene i CMAP-chunken giver efter delingen med 240 de 16 tilladte værdier.

Efter indlæsningen af farvekomponenterne i tre felter, anvendes elementerne i en FOR-løkke til at opbygge den aktuelle PALETTE. Hermed er bearbejdningen af CMAP-chunks'ene færdig.

Så skal den tredje og sidste chunk, BODY-chunk'en, der indeholder de egentlige billeddata, blot bearbejdes. Formatet, dataene er gemt under, er dog et andet end det, som GET frembringer og gemmer i et felt - ellers ville

det være overflødigt at beskæftige sig med IFF. I en BODY-chunk kommer først dataene for første billedlinie i bit-plan 1, derefter dataene for plan 2 osv. til det sidste (femte) bitplan. Så kommer datene for anden linie af det første bit-plan, for anden linie af det andet bit-plan og så fremdeles.

Hermed er opbygningen af billedet forklaret. Hvor skal disse data skal opbevares, for at de kan komme til syne på skærmen? Det sprøgsmål kan ikke besvares generelt, for Amigaen har knapt nok faste adresser, hvori bestemte data ligger. Adresserne, hvor dataene for skærmbilledet står, kan til enhver tid ændres. Heldigvis er der metoder til at konstatere adresserne med udgangspunkt i et vindue. Det sker med WINDOW-funktionen. WINDOW(8) giver adressen på det aktuelle vindues såkaldte "rastport".

adpointer = WINDOW(8)

Rastporten er en kompliceret datastruktur med en hel række felter, der mestendels er uden interesse for det foreliggende program. Det andet felt i rastporten (som begynder 4 bytes bag den første adresse) indeholder adressen på endnu en datastruktur, en såkaldt BitMap.

andadresse = PEEKL(adpointer + 4)

I en BitMap-datasætning kommer først to langord med data, der er uden interesse her, og så (8 bytes efter strukturens startadresse) listen med adresserne på de enkelte bitplaner.

adresse = andadresse + 8

Det andet bitplans adresse ligger fire bytes længere fremme (på andadresse + 12). Dermed er målet nået: man finder adressen på det første bitplan med følgende fire sætninger:

```
adpointer = WINDOW(8)
andadresse = PEEKL(adpointer + 4)
adresse = andadresse + 8
planadresse = PEEKL(adresse)
```

Det kan sammenfattes således:

```
FOR i = 0 TO 4 STEP 4
PRINT PEEKL (PEEKL (WINDOW(8) +4) +8 +1)
NEXT
```

Man får to tal, det andet er 20480 højere end det første. Det er startplanerne for de to bitplaner af den aktuelle skærm.

Forskellen på de to startadresser skyldes antallet af bytes, hvert plan kræver.

Bredde 640 * højde 256 / 8 bits = 20480 bytes

(De to bitplaner behøver ikke at følge lige efter hinanden; deres afstand kan være større. Det er sjældent tilfældet, da Amigaen lægger de nødvendige bitplaner lige efter hinanden, når den åbner en ny skærm.) Nu oprettes den tredje rutine *bildwerte til læsning af IFF-filer. Den indlæser BODY-chunk'ens data direkte i skærmens bitplaner.

REM Teil3

Først deles billedbredden med 8, og resultatet gemmes i variablen bytprozeile-*bytesprlinie (antallet af bytes i en ét punkt høj linie af skærmen).Så hentes adressen på det første bitplan af den aktuelle skærm i variablen adresse&. Den ydre FOR-løkke gennemløber så linierne (*bildhoehe) af billedet, den midterste de enkelte bitplaner (*ebenen) og den indre skriver dataene for en linie af et bitplan fra chunk'en i hukommelsen.

Lad os se, hvad det færdige program kan. Tilføj programafsnittene *Teil1, *Teil2- og *Teil3 og gem resultatet under navnet *Bildlesen. Start programmet, og det beder om et filnavn. Indtast så det følgende navn (elller rutenavnet for et andet billede):

GrafiCraft:Mona Lisa

Fint, ikke? Billedet af Mona Lisa ruller langsomt, linie for linie, ned fra den øverste billedkant. Farverne er som på GrafiCraft-diskettens original (det har indlæsningen med CMAP-chunks sørget for). Prøv med nogle andre billeder.

3.3.3 Gemning i INTERCHANGEABLE FILE FORMAT

Nu er det ikke vanskeligt at skrive et program, der frembringer data i IFF-standard. Hertil behøver man kun at ændre læsekommandoerne (INPUT$) i de første tre dele af programmet til skrivekommandoer (PRINT). Programmet følger i næste listning under overskriften *Teil4, hvilket antyder, hvad det skal bruges til. Det skal senere tilføjes bag *Teil3 af programmet til indlæsning af IFF-filer.

REM Teil4

Rutinen anvender navnet "*Namenlos, når der ikke er angivet filnavn. Det gør programmet alsidigt, så det også kan anvendes til programmer uden navn.

I første afsnit tildeles værdier til de variabler, som kræves til de enkelte chunks. Data for billedbredde, billedhøjde og antal bitplaner, fastslås med WINDOW-funktionen i subrutinen *fensterholen. Antallet af bytes i grafikbilledet beregnes i variablen bytes ud fra billedbredden br, billeddhøjden *hoehe og antallet af bitplaner *ebenen1. Variablen *zus indeholder det antal bytes, der herudover skal skrives til grafikbilledets bytes til grundelementet FORM: 52 (idet der ses bort fra CMAP). Variablen *farbwerte anvendes som pointer, der skal sættes af det program, som kalder *speichernals. Hvis pointeren er logisk sand, adderes tillige længden af CMAP-chunken til *zus. Længden afhænger af antal bitplaner. Derefter kaldes subrutinen *dimen. Hvis pointeren dimension er falsk, bliver feltvariablerne rød, grøn og blå dimensioneret og dimension sat lig -1 (logisk sand).

Nu kan vi åbne filen filename$. Først skrives grundelementet FORM i filen. Så følger BMDH-chunk'en. Først skrives chunk-navnet "BMDH" og chunk-længden 20 i filen, derefter værdierne for billedbrede br, billedhøjde *hoehe osv.

I programdelen cmap aflæses pointeren *farbwerte. Hvis Basic-programmet er forberedt på denne IFF-læse- og skrive-rutine, hentes farveværdierne i hovedprogrammet fra felterne med de enkelte farvers røde, grønne og blå komponenter og udlæses (ganget med 240, se ovenfor). Hvis *farbwerte er falsk (=0), overspringes rutinen cmap, der skrives altså ingen CMAP-chunk. I så fald ændres paletten ikke ved senere indlæsning af filen.

Dernæst kommer BODY-chunken. Først skrives navnet "BODY" og chunklængden (i bytes) i filen. Bitplanernes adresser fastslås igen - som i del 3 - med WINDOW(8). Den yderste FOR-løkke gennemløber linierne i billedet, den mellemste bitplanerne, og den inderste skriver en linie af et bitplan i filen i 4-bytes bidder. I den yderste FOR-løkke udsendes en signallyd for hver gemt linie. Således ved man, hvornår gemningen er forbi. Hvis lyden generer, kan SOUND-kommandoen udelades. Endelig skrives en CAMG-chunk, der er en kopi af samme chunk i Mona Lisa-filen, og den lukkes.

Hermed har vi to komplette utilities til læsning og skrivning af IFF-grafikfiler. De kan også anvendes hver for sig eller med et lille forprogram som ét helt program:

REM iffLesenSpeichern
'P.3.3-6

Gem programmet under navnet *iffLesenSpeichern. Dermed har man et alsidigt program til forskellig brug; det er konstrueret til at kunne anvendes i næsten alle programmer.

Tilføj derefter *iffLesenSpeichern til tegneprogrammet PAINTAmiga fra sidste kapitel. PAINTAmigas sidste tre linier

*Einlesen:...
*speichernals...
*speichern...

skal sætttes ud af kraft med REM-kommandoer eller slettes og iffLesenSpeichern-*ifflaesegemme så tilføjes. Når man derefter bruger PAINTAmiga, bliver billederne automatisk gemt og indlæst i IFF-format. Det har man måske ikke så meget gavn af, men de kan også straks gemmes på en GraphiCraft-diskette. (Filnavn: "GraphiCraft: avn".) Med GraphiCraft kan man så lade billedet udprinte i farver. Omvendt kan man indlæse et GraphiCraft-billede og arbejde videre på det med PAINTAmiga. De to programmer kan så udveksle data (kompatible)! Bemærk, at gamle billeder, der er konstrueret før denne ændring af PAINTAmiga, ikke længere kan indlæses.

Programmet *iffLesenSpeichern kan også gemme delbilleder, som kan prøve med programmet Adventure fra kapitel 5.1. Det kan bruges til at virkeliggøre læserens egne ideer.

3.4 Sekventielle filer

Sidste kapitel gik på filoperationer (og PEEK/POKE) til grafiske anvendelser. Nu skal vi se på andre filoperationer. Dette afsnit behandler de grundlæggende ind- og udlæsningsmuligheder for sekventielle filer.

Amiga-Basic muliggør to slags filer, sekventielle filer og filer med direkte tilgang. En sekventiel fil er let at programmere, men har nogle mangler. Dataene opbevares efter hinanden på denne type fil. Her er det i reglen ikke en simpel strøm af bytes; men en rækkefølge af "datasæt". (IFF-chunks er f. eks.sådanne datasæt.) Hvis man søger et bestemt datasæt, og det ligger bagest i filen, skal man først indlæse (gennemløbe) hele filen, før man kommer til det søgte sæt. Det er besværligt at ændre dette datasæt; men med små filer, er det ikke et stort problem. Husk dog, at en diskette kan rumme 880 Kbytes data, og dermed også store filer. Hovedanvendelsesområdet for sekventielle filer er derfor små filer, der uden problemer kan indlæses på en gang og rene tekstfiler (der nemlig blot er en strøm af bytes).

Man skelner mellem tre forskellige typer tilgang til sekventielle filer: oprettelse (skrivning), tilføjelse og læsning. Ved åbningen skal man afgøre, hvilken type tilgang man vil anvende. Man kan altså ikke skrive noget i en fil, og samtidig læse fra den. Desuden må man ikke åbne en eksisterende fil blot til skrivning, det vil slette de data, der allerede er til stede. I slutningen af afsnittet er der en liste med de punkter, man skal overholde ved arbejdet med sekventielle filer. De to følgende eksempler kan af pladsmangel ikke gå ind på alle disse forhold. *Kfz filen går derimod eksakt frem efter dem.

Den første tilgangsmåde er oprettelsen/skrivningen af en fil. Basic-kommandoen hertil lyder:

OPEN "navn" FOR OUTPUT AS #filnummer

eller kortere

OPEN "O", #filnummer, "navn"

Herved oprettes en fil med navnet, eller en tilstedeværende åbnes til overskrivning og kan senere tilgås med betegnelsen #filnummer. Et lille eksempel viser det.

Anlegen

Programmet spørger, efter åbningen, om navn og nummer på en videofilm og opbevarer oplysningerne i variablerne tit$ og kas$. Med WRITE#-sætningen skrives begge variabler i filen "video. Så længe der ikke er opgivet E for end, kan man gemme det ene datasæt efter det andet. Så lukkes filen med CLOSE. I samme øjeblik fremkommer en fejlmelding, fordi programdelen menu stadig mangler. Den forklares senere.

Ud over OPEN anvender programmet de to nye kommandoer WRITE og CLOSE.

WRITE #filnummer, liste af udtryk
skriver værdierne af udtryk i filen med det angivne navn
CLOSE #filnummer,#filnummer,...
lukker filerne med de angivne numre, derefter kan der ikke skrives flere data i eller læses fra denne fil

For at tilføje flere data til en allerede bestående fil skal man benytte den anden tilgangsmåde. Filen åbnes så med:

OPEN "navn" FOR APPEND AS #filnummer

eller kortere

OPEN "A", #filnummer , "navn"

Følgende programfragment demonstrerer anvendelsen af kommandoen.

Ergaenzen

Som det ses, ligner tilføjelse til en fil oprettelsen og den påfølgende skrivning, bortset fra selve åbningen. De to variabler tit$ og kas$ overgives denne gang til filen med PRINT#-sætningen. PRINT og WRITE udfører i dette tilfælde nøjagtigt det samme.

Den tredje og sidste tilgangsmåde er læsning af data fra en eksisterende sekventiel fil. Det kræver også en særlig åbningsmåde. Hertil tjener følgende kommando:

OPEN "name" FOR INPUT AS #filnummer
eller kortere
OPEN "I", #filnummer, "navn"

Det følgende programfragment viser, tilgangsmåden i et program.

Einlesen:

Efter filen er åbnet til læsning, indlæses data med en WHILE-løkke, til enden af filen (EOF = End Of File) er nået. De indlæste data opbevares i feltvariablerne tit$ og kas$ og vises med PRINT på skærmen. Til indlæsning bruges kommandoen INPUT, som ved indlæsningen af en IFF-fil.

INPUT #nummer,variabel,variabel,
læser data fra den sekventielle fil med betegnelsen nummmer, og tildeler disse til en eller flere variabler

Vil man have et udprint af disse data, kan man tilføje følgende rutine.

Drucken

For at bruge denne, lidt primitive, filbehandling behøver vi blot at skrive programdelen menu og dimensionere feltvariablerne tit$ og kas$. Det sker i følgende programfragment. Indtast det og tilføj bagved de andre programfragmenter til *Erzeugen, *Anhaengen og *Lesen. Gem det under navnet *Video-datei. Bemærk at programmet i den form maksimalt kan tage 100 datasæt; længere er de tre feltvariabler ikke.

REM Video-datei
'P.3.4-1

Som man ser af programmet, er det let at oprette et simpelt filbehandlings-system.

Følgende skema viser strukturen af et program, der arbejder med en sekventiel fil. Det svarer ikke helt til programmet *Video-datei-, der var lidt forenklet.

-dimensionering af felter
-undersøgelse af, om filen allerede eksisterer
-hvis ikke,
    åbn filen til skrivning
    skriv datasæt med fiktive data ind
    luk filen
-menu til forgrening til de enkelte rutiner
    -nye data
        åbn fil til tilføjelse (APPEND)
        skriv data i filen med PRINT#
        luk filen
    -læs tilstedeværende data
        åbn filen til læsning (INPUT)
        indlæs data med INPUT#
        luk filen

Særligt vigtige i skemaet er de første to skridt: afprøvningen af, om filen allerede er til stede, og hvis ikke så oprettelsen af den. Således undgår man at overskrive en tilstedeværende fil. Vi ser senere i dette kapitel, hvordan man kan se, om der er en fil på disketten.

3.5 Kogebog

Som vi har hørt, bruges sekventielle filer mest til behandling af tekster. I dette afsnit vil vi oprette et program til behandling af mindre tekster. Programmet hedder *Kochbuch, fordi man næppe kan indtaste madopskrifter på en enkelt linie, men det kan også bruges til andre formål.

Vil man benytte programmet aktivt, så bemærk oplysningerne i slutningen af forrige afsnit. Programmet er af pladsgrunde ikke sikret mod overskrivelse. Det forekom mig vigtigere at vise, at det heller ikke er noget problem at udlæse formatterede tekster i Amiga-Basic. Før man begynder, bør man anlægge en tom skuffe med navnet OPSKRIFTER på disketten BFA. Den skal optage de opskrifter, programmet behandler.

Men først listningen af programmet *Kochbuch. Forklaringer følger.

REM Koch-datei
REM

Begyndelsen af programmet indtil label'en *ein er velkendt. *Video-datei-er ikke anderledes; men så bliver det interessant. Opskrifterne, som skal gemmes, vil givet være af forskellig længde. Derfor opbevares hver opskrift i sin egen fil. For at kunne finde den igen, gemmes en liste over alle opskrifterne i en ekstra fil. Filen får navnet *koch.

I progammet bag label'en *anlegen oprettes filen "*koch" og så kaldes subrutinen *ein. Den indlæser et navn og den tilhørende opskrift. Der spørges efter opskriftnavnet med sætningen LINE INPUT og strengvariablen *ger$ overgives.

> **LINE INPUT;"tekst"; strengvariabel**
> indlæser max. 255 tegn fra tastaturet og tildeler dem til variablen. Man kan også indtaste kommaer og tomme mellemrum med LINE INPUT.

Det opskriftnavn, indlæst med LINE INPUT, bliver samtidig filnavn for filen, der indeholder opskriften, og bliver senere noteret i filen "*koch". Så indlæses opskriften og hver gang der trykkes en tast (det sørger subrutinen *taste), tilføjer programmet bogstavet til enden af teksten med variablen tx$. Variablen lg tæller antallet af indlæste bogstaver.

Næsten alle bogstaver gemmes, som de er indtastet. Undtagelser er BACKSPACE og RETURN. Opdages et BACKSPACE (=CHR$(8)), så gemmes denne tast først, men derefter slettes de sidste to tegn i tx$ og således ikke kun BACKSPACE, men tillige bogstavet foran, fra teksten. BACKSPACE har altså sin sædvanlige funktion. Såsnart der trykkes RETURN, afsluttes indtastningen og teksten med opskriftens navn gemmes som fil. Variablen lg, tjener til længdebestemmelse af det datasæt, der skal gemmes.

Funktionen CHR$ undersøger om der er indtastet et BACKSPACE.

> **v$=CHR$(tal)**
> giver bogstavet med ASCII-koden tal (melllem 0 og 255)

Til afskæring af de sidste to bogstaver bruges funktionen LEFT$

> **v$=LEFT$(streng,N)**
> leverer en tekst, som består af de første N bogstaver af streng

Et eksempel: indtast de to kommandoer i direkte funtionsmåde:

A$ = "Basic-program":PRINT LEFT$(A$,5)

og der fås følgende output (de første fem bogstaver af A$)

Basic

Subrutinen *holen indlæser en del af en gemt tekst. Den viser listen over disponible opskrifter (via et kald til *inhzeig) og spørger om navnet på den ønskede opskrift. Navnet indlæses så skuffen *Rezepte dermed udvides, og så åbnes filen med opskriften. Indholdet indlæses med LINE INPUT i variablen tx$, og så slettes elementerne i feltet text$, der senere skal optage opskriftens formatterede tekst. Herigennem forhindres, at resterne af den sidst indlæste opskrift vises.
NU kommer vi til den smukkeste rutine i hele programmet:*Format. Her formatteres teksten med fast venstrekant og 60 tegn bred. Først skal vi have antallet af tegn i den tekst, tx$, der skal formatteres.

v = LEN(x$)
giver længden af tegnstrengen x$

Atter et eksempel herpå:
Skriver man følgende sætninger i direkte funktionsmåde

A$ = "tegnsaet":PRINT LEN (A$)
får man tallet 8 som resultat.

Format skal opdele teksten i tx$ i afsnit, der højst er på 60 tegn. Når tekstens længde er mindre end 61, er format allerede færdig. Ellers forsøges at dele teksten på den 60. plads. Dertil sættes variablen br til 60 tegn. Den følgende WHILE-løkke søger efter et mellemrum før den 60. plads og opfanger alle tegn af teksten, der står foran det i et element af feltet text$. Senere slettes så alle disse tegn fra text$. Procesen gentages til text$ er tom. Variablen *zl opbevarer permanent, hvor mange linier, der er oprettet på den måde. Ved opdeling af opskrifter i korte linier anvendes en række nye funktioner, der skal forklares.

v$ = MID$(streng,n,m)
giver den del af streng, der begynder ved position n og er m tegn lang.

Et eksempel på det:
Hvis man indtaster de to følgende sætninger i direkte funktionsmåde

A$ = "Diskettestation":PRINT MID$(A$,4,5)
skriver Basic
kette

Til afskæring af den forreste del af opskriften (i tx$) anvendes funktionen RIGHT$, der giver den højre del af en streng.

v$ = RIGHT$(streng,n)
giver en tekst, der indeholder de sidste n tegn af streng

Igen et eksempel:
Hvis man indtaster de to følgende sætninger i direkte funktionsmåde

A$ = "skrueblyant": PRINT RIGHT$(A$,6)

får man

blyant

Sidst skal der åbnes en mulighed for at få opskrifterne printet ud. Det kan gøres direkte med LPRINT. Først indlæses opskriften med *holen og formatteres i 60 tegns bredde. Navnet understreges i anden linie med en række asterisker (*). Hertil finder programmet antallet af tegn i navnet med LEN og danner med STRING$-funktionen en tekst med tilsvarende mange asterisker i variablen srn$-*ast$

v$ = STRING$(n,tal)
v$ = STRING$(n,streng)
giver en tekst, der består af n tegn med ASCII-koden tal, eller en tekst, der består af n gentagelser af streng

Og naturligvis atter et eksempel

Hvis man skriver de følgende to sætninger i direkte funktionsmåde

A$ = "STRING$(5,7):PRINT A$

hører man fem pippelyde (tegnet med ASCII-koden 7 er ikke synligt på skærmen, men frembringer et pip ved udlæsningen).
Og hvis man indtaster de to følgende sætninger i direkte funktionsmåde

A$ = "mus":PRINT STRING$(10,a$)
så får man
MMMMMMMMMM

Bemærk advarslen i slutningen af kapitel 3.4 for ikke fejlagtigt at slette tegn. Et par forslag til forbedringer: menuen kan også udføres som pull-down menu, som lært med tegneprogrammet PAINTAmiga. Ved lægnere tekster afbrydes udprintningen ved slutningen af siden og der foretages en sidenummerering.

3.6 Talsalat

Har man mange data gemt i en fil, vil man have dem ordentligt sorteret. Man bør vide noget om grundelementerne i sortering for at kunne anvende dem rigtigt.

Til sorteringsforsøg nøjes vi med et eendimensionalt felt på 100 elementer. Vi begynder med at tildele elementerne i feltvariablen a() tallene fra 300 til 100 og med et mellemrum på to. For at gøre det sværere, fordeler vi fire værdier, der ligger udenfor området.

REM sort
REM P.3.6-1

Feltet ser altså således ud før sortering:

300 298 296 294 292 290 288 286 284 282 1111 278 276 274 osv.

Det sorteres opadstigende, og Amigaen tager selv tid derpå.

```
v1$= TIME$
GOSUB sorter1
v2$=TIME$
WIDTH 60
FOR s1 = 0 TO 100
PRINT a(s1);
NEXT s1
GOSUB tid
PRINT
PRINT "Sorteringen tog";T7;"sekunder"
END
```

I v1$ lagres den aktuelle tid før den egentlige sorteringsproces (i subrutinen *sortieren1) og i v2$ den nye tid efter. Subrutinen *zeit tager differensen og lægger den i T7. Funktionen TIME$ fastslår tiden.

v$ = TIME$
viser tiden som tegnstreng af formen "hh.mm.ss" (hvor "hh" er timer, "mm" minutter og "ss" sekunder).

I rutinen *zeit findes de numeriske værdier af sekunder og minutter herfra. Det sker ved at ekstrahere delstrengene med MID$ og RIGHT$ og forvandle dem til tal med VAL.

v = VAL(streng)
giver værdien af streng opfattet som tal

Talllene viser forskellen mellem v1$ og v2$ og viser tiden, som blev brugt. (Læg mærke til, hvordan det klares, når der ligger et timeskift mellem de to tidspunkter.)

zeit

Den første sorteringsrutine *sortieren1 er enkel. Der sammenlignes to nabotal. Er det første tal størst, ombyttes de, så det største kommer en plads længere bagud. Mere formelt: programmet sammenligner a(0) med a(1), a(1) med a(2), a(2) med a(3) osv., indtil det største tal står sidst. Så følger anden gennemgang, hvor næststørste tal kommer på næstsidste plads. (Det kaldes også boblesortering, fordi de store tal stiger opad som bobler.) Ændring af talfeltet kan vises med et eksempel. Heri er gennemløbene af den indre FOR-NEXT løkke i rutinen *sortieren1 angivet med en asterisk (*).

1. Gennemløb af den ydre løkke:

1* 298 300 296 294 292 290 288 286 282 1111...
2* 298 296 300 294 292 290 288 286 282 1111...
3* 298 296 294 300 292 290 288 284 282 1111...
11* 298 296 294 292 290 288 286 284 282 300 278 1111...
...
100* 298 296 294...106 104 102 100 1111

2. Gennemløb af den ydre løkke

...
100* 296 294 292...104 102 100 911 1111

3. Gennemløb af den ydre løkke:

...
100* 294 292 290... 102 100 300 911 1111

100. Gennemløb af den ydre løkke:

...
100* 25 100 102... 296 298 300 911 1111

Se selve subrutinen *sortieren1, der realiserer sorteringsprincipet i Basic:

sortieren1

Første sortering kræver ca. 60 til 61 sekunder til det forudanlagte felt a, unægtelig noget længe. Derfor et nyt forsøg. Vi finder det mindste tal i feltet a og noterer tallet i variablen b og positionen af elementet i variablen c. Elementerne under position c skubbes et trin opad, så nederste position bliver fri. Her anbringes tallet b (det mindste element). I anden løkkegennemgang gennemsøges feltet kun fra det andet efter det næstmindste element - det mindste er jo fundet - og det anbringes på den anden plads i feltet. Processen gentages, indtil sidste element er gennemsøgt.

1. Gennemgang af den ydre løkke:

1* 298 300 296 294 292...
2* 296 300 298 294 292...
3* 294 300 298 296 292...
...
100* 2 300 298 296... 108 106 104 102 100

2. Gennemgang af den ydre løkke:

...
100* 25 300 298... 110 108 106 104 102 100

3. Gennemgang af den ydre løkke:

...
100* 25 100 300 298.. 110 108 106 104 102

100. Gennemgang af den ydre løkke:

...
100* 25 100 102 104... 298 300 911 1111

Rutine *sortieren2, ser således ud:

sortieren2

Husk at linien GOSUB *sortieren1 i hoveprogrammet skal ændres til GOSUB *sortieren2. Man vil så se, at sorteringen af samme felt a, nu blot kræver 27 - 28 sekunder.

Men det er også for meget. Halverer tiden igen. Det gør subrutinen *sortieren3. Det er en lille modificering af *sortieren2. Den anden indvendige FOR-NEXT løkke var her meget tidskrævende; men den er overflødig, fordi det ikke er nødvendigt at skubbe de ikke sorterede feltelementer en position opad. Det er nok at hente det største element i den ikke sorterede del forrest og at sætte det element, der blev fortrængt fra førstepladsen, på dets position. Den subrutine, der kommer ud af det, ser sådan ud:

sortieren3

Og det passer! Subrutinen *sortieren3 klarer sorteringsprocessen på 14-15 sekunder. I det næste afsnit anvendes den i praksis.

3.7 Den der arbejder, gør også fejl

Ved programmering er det vigtigt at tage hensyn til mulige fejl. Et program må ikke kunne crashe på grund af en fejlindtastning. Omkostningerne er for store ved at tage hensyn til alle tænkelige fejl, men det lønner sig at tage højde for de mest almindelige.

AmigaBasic er til stor hjælp ved fejlbehandling. Sætningerne ON ERROR, ERROR, RESUME og systemvariablerne ERR og ERL gør det let at behandle fejl. Det tydeliggøres med et eksempel:

Rem Fehler
REM P.3.7-1

Bliv ikke forbavset over linienumrene. Selv om de normalt ikke er nødvendige, bruges de faktisk til noget her. Vi vil her demonstrere, at Amiga-Basic er ligeglad med, om og hvordan linierne er nummereret, og for det andet kræver disse programmer linienumre til en vigtig funktion. Men det er vigtigt, at ingen af linienumrene forekommer mere end én gang.

I programmets første linie står ON ERROR-sætningen

> **ON ERROR GOTO hopmærke**
> Bevirker, at der gås til hopmærket, når der optræder en programfejl

I modsætning til interrupt-events (ON BREAK o.l.) behøver denne sætningen ingen ekstra aktivering (BREAK ON). Såsnart der bliver fundet en fejl, forgrener programmet til det ønskede sted uden at udsende fejlmelding. Dermed kan reageres på alle fejl, man kan forudse. Et specialtilfælde af denne sætning er ON ERROR GOTO 0, der sætter den normale fejlbehandling i kraft igen.

De følgende linier 2 til 10 er ikke så interessante, programmet beder om værdier af vej og tid. Men pas så på, den næste linie simulerer en fejl. Ja, det passer, man kan lade en særlig fejlmelding udlæse for hvert enkelt programtrin.

ERROR nummer
simulerer en programfejl med fejlkoden nummer

I sætningen kan anvendes både egne og de af Basic definerede fejlkoder (de findes i manualen). De første skal ligge uden for standard-fejlkoderne (f.eks. over 150). I vort eksempel defineres fejlkoden 151, når den indtastede tid er mindre end 0 minutter eller større end 24 timer. Prøv at starte programmet og angive tiden -25 som tidsværdi. Programmet melder sig med meldingen "Tiden udenfor det gyldige område".

Betragt nu programmet fra linie 20, dvs. den linie, hvor der forgrenes til fejlbehandlingen (ON ERROR GOTO 20). Der spørges først, om systemvariablen ERL=13.

v=ERL
giver nummeret på den linie, hvori den sidste fejl optrådte

Skal variabelen kunne udnyttes, skal der anvendes linienumre; kun således kan man lokalisere fejlen præcist. Hvis der er en fejl i linie 13, hopper programmet til linie 25. Hvad sker der i linie 13? Vi beregner hastigheden ved at dele vejlængden med tiden. Indtastes 0 som værdi for tiden, deles i den følgende formel med 0, og normalt ville man få fejlmeldingen Division by zero, og programmet ville blive afbrudt. Programeksemplet forgrener i stedet til linie 25 og udlæser meldingen "Division med nul er ikke tilladt". Efter en kort tidsløkke, der giverlejlighed til at læse teksten, finder man sætningen RESUME.

RESUME hopmærke
fortsætter et program efter hopmærket

Programmet begynder derfor forfra i linie 2, uden at være blevet afbrudt, som det ellers sker ved den slags fejl. Prøv engang. Fejlene opfanges i linie 22, og de fejlkoder, der ikke er forudset i programmet, udlæses her.

v=ERR
giver den sidst optrådte fejls fejlkoder.

En kontrol af ERR bør foretages i alle fejlbehandlingsrutiner. Det er bedre at afbryde et program ved uforudsete fejl, før det kommer til komplikationer.

3.8 Mine biler

Vi vil nu prøve anvendelsen af sekventielle filer, på et noget større projekt. En slags bogføring af omkostningerne ved at holde bil.

Det er praktisk at vide, hvor meget, det koster at holde bil. Der findes bøger, hvori man kan indføre den slags omkostninger gruppevis og få resultatet med møjsommelige beregninger. Vi vil lade Amigaen klare det, eller rettere et Basic-program.

Også flere biler kan programmet klare. Dataene kan selvfølgelig udlæses sorteret. Amigaen har fremragende grafik, så vi kan vise tallene som letforståelig grafik.

Vi begynder med at aktivere en fejlbehandlingsrutine i programmet. Efter dimensionering til 200 datasæt (kan ændres efter behov) sættes de to pointere *offen og *sort til 0 (logisk falsk). Når den valgte bilfil behøves, viser pointeren *offen, om den allerede er indlæst. *sort fungerer ligesådan, den viser, om den aktuelle fil er sorteret. Indtast så den følgende listning. (Husk på, at linier med to asterisker (**) hører til den foregående linie og skal indtastes på en enkelt linie sammen med denne.)

REM KFZ-datei
REM P.3.8-1

Derefter følger ved hopmærket *start åbningen af bilfilen "*name" til indlæsning. I filen "*name" gemmes de biler, der skal føres bog over. Når programmet kører første gang, er der selvfølgelig ingen fil. OPEN-kommandoen giver så en fejl, hvorved programmet hopper til rutinen *fehler og derfra til *anlegen. Der oprettes filen "*name" og navnene på bilerne indlæses i en WHILE-løkke. Variablen j fungerer som tæller for antallet af navne. Er der ikke gemt noget navn (j, opfordres man til at indtaste et. Findes kun et navn (j, forgrenes der til menu. Er der flere navne, og dermed filer, oprettes en ekstra valgmenu hertil.

start

Når programmet kommer til hopmærket *menue, kaldes subrutinen *menuzeigen-, som viser de til rådighed stående menuer. Alt efter indtastningen forgrenes der til den ønskede funktion. Ved *ende afprøves, om der er foretaget en ændring. I så fald gemmer subrutinen *speichern de ændrede data på diskette.

menue
menuezeigen

Vil man kun indføre et enkelt datasæt, bør det, som altid gå så hurtigt som muligt. Derfor er der to rutiner til indførelse. Den første, *eintragen, er beregnet på den nævnte hurtige indførelse. Filen åbnes til tilføjelse (APPEND), og de nye data tilføjes bag de gamle. Så hoppes tilbage til menuen, hvorfra brugeren kan afslutte programmet. Hvis man arbejder længere tid med programmet, og filen allerede er indlæst og står til rådighed i de dertil beregnede feltvariabler (se nedenfor), så gemmes det nye datasæt i disse felter ved hjælp af rutinen *eintragenoffen, og skrives ikke på disketten med det samme. Den egentlige indtastning af værdierne sker i subrutinen *eingabe, der beskrives længere fremme.

eintragen
eintragenoffen

Vi kommer nu til subrutinen *anlegen, der blev omtalt under fejlbehandling,og heri oprettes filen "*name", der indeholder de biler, der skal føres bog over. Filen oprettes og åbnes som fil#2 (#1 bruges til omkostningsdataene). Ser man nøjere på filen, vil man opdage, at der ved oprettelsen bare foretages en pseudo-indførelse, der senere bliver den første linie til menuudlæsning (se nedenfor). Den faktiske indførelse af den første bil, og alle følgende, sker i tilføjelses-mode.

anlegen
ergaenzen
ersteintrag

Sletningen af et datasæt er ikke vanskelig. Der bedes om nummeret på datasættet, der skal slettes. Kender brugeren ikke nummeret, kan han forlade funktionen og finde ud af det. Sletningen sker så ikke direkte i filen, men i de feltvariabler, der optager datasættene. Eventuelt indlæses hele filen. Programmet henter så hele tiden det følgende datasæt i en FOR-løkke og gemmer det et trin længere nede, idet det begynder ved det sæt, der skal slettes. Først bliver den variabel, der indeholder antallet af sæt, formindsket med én.

loeschen
loeschenoffen

Nu følger indtastningen af et datasæt. De første fire indtastninger er tvungne, dvs. programmet læser først den femte indtastning, når de fire er foretaget. Det forhold kan man dog til enhver tid lave om på i programmet. Den fjerde indførelse (kilometerstand) er ubetinget nødvendig, fordi den benyttes til sortering af dataene. Programmet tager det ikke så nøje med resten af indførelserne. Hvis brugeren kun trykker RETURN, tildeles den tilsvarende variabel blot et nul.

Eingabe

De to følgende rutiner *speichern og *einlesen, er bekendt fra de foregående afsnit.

speichern
einlesen
lesen
lesenoffen

Programdelen *lesen står for visning af de enkelte datasæt på skærmen. Efter kontrol af de to pointere *offen og *sort og eventuelt indlæsning af filen i feltvariablerne, vises en overskrift. Herunder defineres tegnstrengsvariablen a$ som bestående af to tomme mellemrum ved hjælp af funktionen SPACE$.

> **v$ = SPACE$(antal)**
> giver en streng på antal mellemrum

Herefter udlæses dataene formatteret. Den mest alsidige sætning til formatteret skærmudlæsning er PRINT USING.

> **PRINT USING format;liste**
> viser udtrykkene i liste i format på skærmen

Der er så mange muligheder for at formattere tekster og tal med format, at det kunne fylde et helt kapitel; men det kan man se i Basic-manualen. Her nøjes vi med eksemplerne i programmerne.

PRINT USING-sætningerne står i en WHILE-løkke, der viser en ny linie på skærmen efter hvert tryk på mellemrumstangenten. Først vises sætnummeret (s) med tre cifre (Format "###"). Har tallet under tre cifre, vises de med lige højrekant. (Hvis det har mere end tre cifre, bliver formatteringen ugyldig, og alle cifrene vises med et %-tegn foran.)

Dag og måned anvender styretegnet "##". Her ligesom før (dvs. tocifret udlæsning af tallet), blot vises et punktum efter værdien. Efter variablen **vb-fb (forbrug pr. liter) er der endnu en formatteringsvariant. Styretegnet "##.##" udlæser to cifre før og efter decimalpunktummet. Hvis der er flere decimaler, afrundes til to. Er værdien mindre end 1, skrives et nul foran decimaltegnet.

Når alle værdier i filen er vist på skærmen, fremkommer teksten til afslutning af indførelsen efter et tryk på mellemrumstangenten (CHR$(32)). Efter et tryk mere, hopper programmet tilbage til menuen.

Det er ikke så nemt at ændre et datasæt, men ved et tryk på RETURN kan man se de gamle værdier under indtastningen af de nye.

Først spørges om nummeret på det datasæt, der skal ændres (*Saz). Derefter vises samme tekst som ved indførelsen, men lige bag den feltets øjeblikkelige værdi. Med LOCATE sættes cursoren på begyndelsen af den gamle feltværdi, og der opfordres til at overskrive denne værdi. Så indlæses den indtastede tekst med LINE INPUT. Trykkes RETURN uden at skrive noget, overgives en nulstreng ("") til LINE INPUT, som afprøves, og den tomme indtastning erstattes af den gamle feltværdi (der i mellemtiden har været opbevaret i *merk$.)

aendern
aendernoffen

Den første programdel, der ikke kun gemmer og udlæser data, men også benytter dem, er *verbrauchswerte. Denne subrutine beregner det samlede kilometerantal for hvert år. Summerer brugte antal liter brændstof og udgifterne hertil, og adderer tillige alle andre omkostninger.

verbrauchswerte
verbrauchswerteoffen
DLF

Så kommer konstruktionen af et diagram, der viser de beregnede værdier i overskuelig form. Først udlæses teksterne, og der tegnes et gradnet. De enkelte årsværdier, der er beregnet af *verbrauchswerte summeres og beregnes til en samlet værdi. En FOR-løkke udlæser værdierne på skærmen og tegner forbrugskurven. Til sidst tegnes gennemsnitsforbruget som små røde rektangler (som en tyk punkteret linie). Et tryk på mellemrumstangenten bringer brugeren tilbage til menuen.

WTR

Så er programmet stort set færdigt, vi skal blot forklare nogle hjælperutiner. Underrutinen *zeigen er indrettet sådan, at den kan bruges til at vise begge de anvendte menuer. Først viser den de to overskrifter tit$(0) og tit$(1) i et rødt rektangel, og en FOR-løkke udlæser indtil ni menupunkter under hinanden. Derunder ses lidt forskudt "vælg med mellemrumstangent".

Efter skift af tegnefarve, slettes området til venstre for menupunkterne, og der tegnes en rød trekant til venstre for første menupunkt ved hjælp af AREA og AREAFILL. Medens programmet venter på en tastaturindlæsning, hopper det hele tiden til linie 5 i begyndelsen af subrutinen. Henvisningspilen slettes og tegnes således hele tiden og blinker derfor. På grund af farveskiftet i programforløbet skifter farven også hele tiden på teksten vælg med mellemrumstangent.

Trykker nu på mellemrumstangenten, hoppes der til subrutinen *pfeil. Her forhøjes variablen y1 med 16 (det næste menupunkt er valgt). Henvisningspilen hopper også hen ved siden af næste menupunkt. Er sidste menupunkt nået (det fastholdes i variablen y2), sættes y1 tilbage. Når man trykker RETURN (CHR$(13)), vender subrutinen tilbage til det program, der kaldte den (bag hopmærket *menue i begyndelsen af programmet). Det beregner ud fra y1, hvilket menupunkt der er valgt, og kalder så subrutinerne for de forskellige funktioner.

zeigen
fa=fa+2
pfeil

Den sidste rutine er ansvarlig for sorteringen af data. Den er bygget, som den er vist i kapitel 3.6, men virker noget uoverskuelig her, fordi der sorteres flere felter parallelt. Et nærmere eftersyn vil fastslå, at det virkelig er rutinen *sortieren3. Sorteringskriteriet er kilometerstanden, der forvandles til et tal med VAL(km$(s)) - det blev jo gemt som en tekst.

sortieren

Hermed afslutter vi behandlingen af sekventielle filer og vender os mod filer med "direkte tilgang".

3.9 Direkte tilgang - hurtig og alsidig

Dette kapitel går på en type filer, der er mere alsidig anvendelig end de sekventielle. Filer med direkte tilgang giver kortere tilgangstider til vilkårlige datasæt. Når man anvender direkte tilgang, kan man indlæse en tilfældig fil (hvis

nummer man dog skal vide), ændre den og atter gemme den. Det er unødvendigt at læse de foranliggende datasæt, men kun det, man lige har brug for.

Opbygningen af en fil med direkte tilgang kan forekomme kompliceret; men det er forkert. Det er tilstrækkeligt at åbne filen ved programmets begyndelse og til sidst at lukke samtlige filer med CLOSE.

Det er dog ikke lutter lagkage. Hver dataoperation, i en fil med direkte tilgang, kræver et datasæt-nummer. Det er nødvendigt for at kunne tilgå dataene direkte. Det er også derfor vi forfølges af numre i privat- og forretningslivet. Fordi det er svært at huske alle disse numre, bør programmøren helst kun anvende dem i ét program, og indbygge et hjælpemiddel til at forvandle numre til en mere forståelig form.

Den anden mangel, ved filer med direkte tilgang, er en ødslen med hukommelsesplads. Datasættene har altid en fast længde, så Basic let kan finde dem. Den faste længde reserveres til hvert datasæt, ligegyldigt hvor meget, der virkelig udnyttes. Anvenders en fil med sætlængden 1024, bytes og gemmer et datasæt, som kun indeholder 20 bytes, går der 1004 bytes til spilde!

Sætlængden skal angives ved åbningen af en fil med direkte tilgang. Det sker i OPEN-kommandoen, der kun har én udformning til filer med direkte tilgang (fordi der ikke er forskellige tilgangsmåder som ved sekventielle filer).

OPEN"R",#filnummer,"filnavn",datasætlængde

Her kan man ikke udelade datasætlængden som i sekventielle filer. Sammen med datasætnummeret tillader denne længde, at man direkte finder det sted på disketten, hvor et bestemt datasæt er gemt. Opdelingen af dataene inden for et datasæt i enkelte felter sker med FIELD-sætningen:

FIELD #filnr, L1 AS var1, L2 AS var2,..
tildeler dataene i et datasæt i filen med betegnelsen filnr til en række strengvariabler De første L1 bogstaver tildeles til var1, de næste L2 bogstaver til var2 osv. De første L1 bogstaver af sidst læste datasæt befinder sig altid i variablen var1, de næste L2 i var2 osv.

NB: Summen af de enkelte feltlængder må ikke være længere end den i OPEN-sætningen fastlagte datasætlængde.

NB: Tegnstrengsvariablerne fra en FIELD-sætning må ikke tildeles en anden værdi (heller ikke med INPUT)!

Dvs.: efter FIELD-sætningen FIELD#1,5 AS A$,7 AS B$, er de følgende sætninger ikke tilladte (selv om de ikke giver fejl):

```
C$ = A$
INPUT A$
LET D$ = A$
```

Nu anvender vi det, vi lige har lært på et eksempel:

oeffnen
REM program1
schreiben

I rutinen *oeffnen åbnes en fil med direkte tilgang og en datasætlængde på 20 bytes. Datasættene opdeles i to felter med FIELD-sætningen. Det første (fircifrede) felt skal optage en artikelbetegnelse og forbindes til strengvariablen *num$. De øvrige 16 bytes er beregnet på en artikelbetegnelse, der er forbundet til streng-variablen *bez$.

I rutinen *schreiben bedes om nummeret på et datasæt. Nummeret er nødvendigt ved alle tilgange til filen. Så indtastes et datasæt, og tildeles til de to variabler *nu% og *nam$ (ikke *num$ og *bez). Ved hjælp af LSET-sætningen overgives de to variablers data så til FIELD-variablerne *num$ og *bez$.

LSET FIELD-variabel = variabel
lægger værdien af variabel, rettet ind til venstre, ind i Field-variablen, så disse data kan skrives i en fil med direkte tilgang med PUT-sætningen

RSET-sætningen gør noget lignende. Den er kun anført for fuldstændighedens skyld, for i de følgende programmer bruges altid LSET.

RSET FIELD-variabel = variabel
lægger værdien af variabel, rettet ind til højre, ind i FIELD-variablen, så disse data kan skrives i en fil med direkte tilgang med PUT-sætningen

For at numeriske data, som *nu% i eksemplet skal kunne skrives i en fil med direkte tilgang, skal de omdannes til en tegnstreng. Det klares af MKS$(nu%). (Forvandlingen af forskellige talformater blev beskrevet i kapitel 2.7.)

I den næstsidste linie af programeksemplet skrives det således udviklede datasæt med sit nummer på disketten.

PUT #filnummer,#datasætnummer
skriver indholdet af FIELD-variablen som et datasæt i filen med betegnelsen #filnummer

Prøv programmet og skriv nogle datasæt i filen. Det er nødvendigt, for at følgende program, der viser indlæsning af data fra en fil med direkte tilgang, skal have noget at læse.

oeffnen:
lesen:

Der spørges om datasætnummeret, og det bruges til at hente filen fra disketten. Det tjener funktionen GET til.

GET #filnummer,sætnummer
indlæser datasættet med nummeret sætnummer fra filen med betegnelsen filnummer i den tilsvarende FIELD-variabel

De data, der er læst med GET, står nu parat i FIELD-variablerne og kan udlæses med PRINT; vi må huske at genoprette de numeriske variabler. Og da skrivning og læsning af data i almindelighed hører sammen, sammenfatter vi de to programstykker til ét program.

REM Teile
REM P.3.9-1

Hvor der står program1 og program2, skal man tilføje de to første programmer i afsnittet og kan så arbejde med programmet. I subrutinen *oeffnen skal man aktivere RETURN-sætningen ved at slette REM'et foran. Desuden skal linien IF *satz% = 0 THEN END ændres til IF *satz% = 0 THEN start i begge programmer.

Til slut igen de vigtigste skridt i et program, som arbejder med filer med direkte tilgang:
åbning af en fil for direkte tilgang (option "R")
opdeling af datasættet med FIELD-sætningen
skrivning:
indtastning af data
overførelse af dataene til FIELD-variablerne med LSET eller RSET
gemning af dataene på diskette med PUT
læsning:
indlæsning af dataene i FIELD-variablerne med GET
udlæsning eller forarbejdning af dataene fra FIELD-variablerne
lukning af filen med CLOSE

Det er tilstrækkeligt til et første overblik. Nogle ekstra finesser gives i de følgende afsnit.

3.10 Terminskalender

Mange har prøvet at glemme en vigtig dato eller termin. Det kan undgås med følgende program. Det forudsætter, at man gemmer alle sine terminer i programmet. Der er plads nok til posteringer. Man kan indføre 260 tegn for hver dag i året, og hvis man vil, kan man indføre terminer, der er af betydning næste år eller senere.

I sidste kapitel lærte vi, at alle operationer på en fil med direkte tilgang, kræver et datasæt-nummer. Læseren undrer sig måske over, hvordan det undgås i terminskalenderen. Til hver termin hører en datoangivelse. Programmet beder om datoen og omregner den til kalenderdage, der bliver til datasætnumre.

Først en advarsel: programmet anlægger en separat fil for hvert år, man indfører terminer for. Når datoen indtastes, afprøves dens korrekthed (man kan indtaste datoer mellem 1986 og 1999). For hver dag i hvert år reserveres plads i den tilsvarende fil til et datasæt (261 bytes). Hvis man indfører data for 14 forskellige år, udgør det 14 filer på hver 95526 bytes eller ca. 93 Kbytes. Så slår en diskettes kapacitet slet ikke til; men til lagring af terminer ét eller to år forud skulle der være rigelig plads.

En anden vigtig ting er markeringen af tomme datasæt. Det kræver lidt tid, men er nødvendigt for en ordentlig databearbejdning. Derfor markeres hvert datasæt under anlæggelsen af en ny fil, med et specielt tegn som værende tomt (se nedenfor).

```
REM Termin-kalender
REM
```

I programdelen menu står valget mellem forskellige rutiner, og der forgrenes til en af dem. Efter udførelsen af den ønskede funktion, vender subrutinen *menue2 tilbage til start.

```
ende:
datum
datum1:
grundzahl
aufloesung
```

I rutinen *datum, der kaldes lige efter start, spørges om datoen. For at mindske risikoen for indtastningsfejl, afprøves det, om dagen er større end 31 eller mindre end 1, og om måneden er større end 12 eller mindre end 1. Årene begrænses til området 1986-99. (Hvis man vil forhindre, at diskettens kapacitet overskrides, kan man ændre grænserne for årene efter behov.) Ved indtastningsfejl forgrener programmet til begyndelsen af rutinen.

Rutinen *grundzahl gennemløbes én gang ved programforløbets start. Den gemmer i feltvariablen *gz, med hvilken dag regnet fra årets begyndelse de tolv måneder i et år begynder. Det er nødvendigt for at kunne finde nummeret på et datasæt, der hører til, i programafsnittet *aufloesung. For at opnå det adderes dagsangivelsen i datoen til det tilsvarende grundtal (*gz) for den pågældende måned.

SUB kopf STATIC
init:
zur:

I underprogrammet *kopf tegnes programmet titelskærm. Det meste klares af rutinen *init, der desværre er ret langsom. Derfor går *kopf kun til subrutinen *init, når titlen endnu ikke er tegnet (hvad der noteres i variablen *z). Hvis titlen er tegnet, gemmes den med GET, og kan altid sættes tilbage på skærmen med PUT.

pruefen:
fehler:

Ved start, lige i begyndelsen af programmet, forgrenes til subrutinen *pruefen. I den åbnes den sekventielle fil "*jahre", hvori de aktiverede år, som der er indført terminer for (og som findes som filer), skal gemmes. Er filen ikke til stede, forgrenes (via ON ERROR GOTO i programstarten) til rutinen *fehler. Hvis fejlkoden er 53 (File not found), forgrenes der til *anlegen (se nedenfor), hvor der frembringes en fil "*jahre".

Hvis filen "*jahre" allerede eksisterer, og der indtastet en dato i et år, der ikke findes i den, så tilføjes året med APPEND bagest i den bestående fil "*jahre" og "aktiveres" (i rutinen *aktivieren).

anlegen:
aktivieren:
oeffnen:

I rutinen *aktivieren åbnes en ny årsfil, og i hvert datasæt skrives CHR$(0) (bogstavet med ASCII-kode nul) som begyndelsesmarkering. Processen ta-

ger nogle minutter, idet hvert år kun skal markeres én gang. I rutinen *oeffnen, der kaldes af *anlegen, bliver den nye fil både oprettet og åbnet. Årsfilens 261 bytes lange datasæt opdeles samtidig med en FIELD-sætning i ét felt på 1 byte (begyndelsesmarkering) og fire felter på 65 bytes. De fire felter bruges til fødselsdage, påmindelser, terminer og andre informationer vedrørende datoen.

lesen:

Rutine *lesen henter det datasæt, hvis nummer blev beregnet ud fra den indtastede dato, fra disketten over i feltvariablerne, og bringer det med en række PRINT-sætninger på skærmen. Hvis datasættet begynder med CHRS$(0), vises teksten "ingen indførelse til stede". Det er noget sværere at skrive en ny termin på disketten. Hvis et nyt sæt simpelthen skrives med PUT på disketten, slettes de gamle data i sættet. Det ønsker vi ikke; desuden er det ratt at kunne se, hvad der står på datoen, når man indfører en ny termin.

I rutinen *schreiben hentes datasættet med GET fra FIELD-variablerne, ligesom i *lesen. Har sættet endnu ikke været brugt, og er forsynet med et CHRS$(0), fyldes alle felter med 0, og sættet skrives i filen (GOSUB 111), hvor den første byte markeres med CHRS$(1). Så kaldes rutinen *schreiben på ny - og der står ikke mere noget CHR(0) i begyndelsen af datasættet. *schreiben udlæser det gamle indhold af terminen. Nu forsøges at finde et sted i terminskalenderen, der ikke er optaget, ved hjælp af INSTR-funktionen.

> **v=INSTR(start,streng,søg)**
> søger og giver som resultat den position, hvor en søgetekst søg forekommer første gang i en anden tekst streng (hvis start udelades, begynder søgningen ved første bogstav af teksten, ellers ved første position start

Et eksempel: Indtastes den følgende linie i direkte funktionsmåde:

```
a$="Eng 9:b$=9":c=INSTR(a$,b$):IF a THEN ? a ELSE?"ikke fundet"
```

får man : 5

*schreiben søger to tomme mellemrum under den antagelse, at der ikke står tekst efter dem i det undersøgte felt. De første tegn indtil disse mellemrum gemmes i variablen *ar$ og udlæses på skærmen. Man kan indtaste den nye tekst bagved, hvor den lagres sammen med den gamle i en af fire tekstvariabler (*geb$, *eri$, *ter$ og *son$) og lægges ved slutningen af rutinen i FIELD-variablerne, som gemmes med PUT. Det sker en gang for hver af de fire felter i hvert datasæt, fødselsdage, påmindelser, terminer m.v.

4 SPRITES, BOBS OG LYD

Stikord, der lyder som en trylleformel, er tæt forbundet med Amigaens egenskaber, der ikke blot er til spil, men også til alvorlige anvendelser. Musens pil er f.eks. en sprite.

Man kan få computeren til at stille spørgsmål i rigtigt sprog og ikke kun med udlæsning, at gøre opmærksom på fejl. At give mundtlige forklaringer i stedet for meldinger. Amigaen kan det hele. Som amerikaner har den svært ved at tale fremmede sprog, men den kan klare dansk.

De grafiske og musikalske evner kan også bruges mindre seriøst. En passende ledsagemusik til et adventurespil forhøjer virkningen væsentligt. Og hvad ville Star Wars være uden lydeffekter. Man behøver ikke at sidde og glo foran computeren, medens den klarer en større opgave. Den kan kalde med en række toner, når den er færdig. De fire lydkanaler giver muligheder nok.

Alt det og meget mere kan man arrangere med Amiga. Man kan ikke påstå, at alle muligheder er udtømt med dette kapitel, men det grundlæggende, der kan klares med Basic, vil man finde her.

4.1 Hallo - det er Amiga

Det er udsædvanligt, at en computer kan tale, og at det kan klares med Basic, og endda kun kræver en enkelt kommando. Amigaen er amrikaner og de kan i reglen ikke tale dansk, men kommandoen SAY, der nærmest taler internationalt, kan være til hjælp.

> **SAY codes, mode**
> udlæser phonem-koderne i strengen codes som sprog, hvorunder betoningen afhænger af det numeriske felt modes indhold

I manualen står under SAY: med denne kommando kan man få "...forståeligt sprog af enhver nationalitet...". Det passer også, da alle sprog på jorden kan præsenteres med fonetisk skrift. Forudsætningen er, at man har taget hensyn til alle sprogmuligheder, og det har Amiga-sprogsyntesens konstruktører desværre ikke. Prøv at lade den tale kinesisk.

Lad os se på SAY-sætningen, eller rettere phonem-koderne. For at kunne sige noget med SAY, skal man danne en tegnstreng bestående af phonem-koder, der giver teksten. Der er en beskrivelse af disse phonem-koder i et

appendiks til Basic-manualen. Men prøv SAY-sætningen ved at indtaste følgende kommando i direkte funktionsmåde. (Systemet forlanger eventuelt, at Workbench-disketten indlægges.)

SAY "AY AEM AX KAA5MAXDOHR PERSINUL KUMPYUW5TER"

Var det let at gætte, hvad der kom ud af det?

I am a Commodore Personal Computer

Så 'nemt' er det. Det er ikke så let at finde den rigtige phonem-kode til hvert bogstav eller bogstav-kombination. Det har Amiaga-Basics konstruktører nok også indset, for de lavede TRANSLATE-funktionen, der klarer arbejdet for os:

v$ = TRANSLATE$(engelsk tekst)
giver en streng, der indeholder passende phonemer for den opgivne engelske tekst, som altså siger teksten, når den overgives til SAY

Man skal kun overgive den engelske tekst til funktionen, så er man fri for at søge efter phonemer. Prøv med det samme:

REM englisch, translate
'P.4.1-1

Som man ser, fungerer TRANSLATE og SAY også perfekt med længere tekster. Ifølge BASIC-manualen kan man frembringe en phonem-tegnstreng på indtil 32767 tegn med denne funktion; dvs. otte til ni siders maskinskrift. Det vil tage et stykke tid for Amigaen at sige alt det.

Man kan ændre tonefaldet på stemmen efter smag. Måske vil man hellere høre en kvindestemme, eller en monoton robotstemme, en magelig stemmeføring eller en stakkato-stemme, næsten alt er muligt. Det prøver vi med en modifikation af sidste eksempel.

REM englisch2, translate, modus
'P.4.1-2

Mode'erne til SAY-sætningen er i eksemplet tildelt til felterne i heltalsfeltet a%. Vil man udvikle egne programmer, kan de lægges i DATA-sætninger og indlæses i starten af programmet. Det har jeg undladt, for at det skal være lettere at ændre de enkelte værdier. Der er en kommentar ved siden af de enkelte feltelementer, der viser betydningen for udtalen. De forudinstillede værdier,

der gælder, når man ikke angiver nogen modus, ses i spidse parenteser i begyndelsen af kommentaren.

Nu kan der eksperimenteres med de forskellige mode-elementer. Prøv at ændre a%(3) til 1, a%(0) til 250 og a%(4) til 2350. Nu taler en kvindestemme. Indtast de forrige værdier igen og tildel et 1 til a%(1). Teksten lyder monoton, netop passende for en computer. Ved ændringen af stemmekvaliteten skal man gå forsigtigt frem og ikke afvige fra den forudindstillede værdi. Ellers lyder det forfærdeligt. Lavere værdier end det forudindstillede lyder dumpt og hult, og højere højt og metallisk. Hvis man har tilsluttet et stereoanlæg, kan man prøve indstillingen af de forskellige frekvensbånd.

Efter disse eksperimenter med engelsk, nu til tysk. Det lader sig ikke løse med nogle Basic-sætninger. Man må sammensætte de enkelte ord efter phonem-kode tabellen i Basic-manualens appendiks H. Desværre er ikke alle tyske lyde repræsenteret, så man må klare sig med lidt afvigende udtale. Det kan være svært og tidskrævende at aflokke computeren forståelige ord, og en amerikansk accent undgår man ikke.

REM deutsch, modus
'P.4.1-3

Med lidt tilvænning kan man nok forstå, hvad der bliver sagt. Prøv nogle andre værdier for feltet a%, måske forbedrer det sprogkvaliteten. Det er et spørgsmål om øvelse. Hvis man mærker sig de vigtigste lydkombinationer, kan man nå langt.

Prøv en gang, om TRANSLATE-funktionen kan være til nytte, når der tales dansk.

REM deutsch2, translate, modus
'P.4.1-4

Som man hører, lyder teksten nogenlunde forståeligt. Hele fidusen er at forsøge at lave en tekst, der lyder dansk, når den udtales som engelsk (altså forkert) af en amerikaner. Tit er det nok at ændre et enkelt ord i en sætning. Skriv f.eks.følgende sætninger i tekstvariablerne a$ og b$:

> "Bawhrnenh hennterh kowhrnett."
> "Moohrerenh hennterh harhmerenn."

De to sætninger er ret forståelige. Den tredje mulighed har jeg gemt til sidst, fordi den er mere krævende. Den er let at indbygge i egne programmer, og

man behøver ikke at indlægge Workbench-disketten for hvert ord, der skal udtales.

I Amiga-Basic manualen findes, i Appendiks H12 et program til dansk sprogudlæsning. For at kunne anvende programmet skal man indlæse teksten i filen test.txt. Til indbygning i et program er det ikke praktisk. Når man ser på programmets DATA-sætninger, kan man i ånden se sine fejlanslag. Dog klares det nemt.

Programmet ligger nemlig også i Demos-skuffen på Amiga-Basic disketten. Navnet på programmet er speechd. Tag en kopi af programmet og læg den uden for Demos-skuffen. Lav navnet om til "dansk", og åbn ikonen med et dobbeltklik. Nu er vi atter i Basic. Der vises en requester, som beder om indlægning af Workbench-disketten. Det gør vi imidlertid ikke, men klikker i stedet i feltet cancel. Derpå kommer Basic med fejlmeldingen Illegal function call, og indrammer en programlinie med rødt. Klik fejlmeldingens OK-felt, og så engang til i listningen, så den røde indramning forsvinder. Nu ligger programmet på skærmen, parat til bearbejdning.
Nu slettes den del af programmet, der vedrører tastaturindlæsning. For ikke at slette for meget, gåes langsomt frem. Samtidig skal man sammenligne med Appendiks H12 i Basic-manualen for at se, om man er nået til begyndelsen af rest-listningerne. Slet nu, forfra i programmet rutinen *start. Så følger rutinerne hand og DrawControls. De sidste to programdele, som skal slettes, er HandleMouse og InitSpeech. Nu har vi næsten klaret det. Den første programdel, man behøver, er initdansk.

PAS PÅ!! *Programmet må ikke gemmes endnu! Der er nemlig en alvorlig fejl i programmet; gemmer man det og loader det igen, virker det pludselig ikke mere.* Det tog lang tid at finde fejlen. I DATA-sætningerne anvendes nemlig også danske specialbogstaver (æ, ø, å), og det tåler Amiga-Basic dårligt, hvad programmøren har overset. Fejlen, er let at rette. Slet den syvende linie fra neden i DATA-sætningen phonemer, som ser således ud:

```
DATA "",2 "H",ER "",ER
```

Så kan vi lagre hoveddelen af programmet (som ikke er på disketten af ophavsretmæssige grunde). Når der i følgende programdel står "Indsæt restprogrammet dansk her", menes denne del af programmet. Den kan ganske vist ikke køre alene. Her er et kort program, der klarer det.

REM Sprache
'P.4.1-5

Så er programmet køreklart. Før afprøvning er det bedst at gemme det under navnet "dansk". For at demonstrere, hvordan man udnytter det i egne anvendelser, følger et regneprogram til de fire grundlæggende regnearter. For ikke at blive for omfangsrigt, regner det kun med heltal (med endelsen %). Vent ikke for meget, vær glad for, at Amigaen taler - trods alle mangler.

REM sprechende Rechenmaschine
'P.4.1-6

Til at begynde med anvendes den kendte ON ERROR-sætning. Rutinen *rech opdeler den indtastede formel i sine bestanddele. Variablen *ze% får det venstre tal, *ze3$ regnemåden og *ze4% det højre tal. Resultatet *erg% beregnes i en IF-løkke, og alt efter regnearten tildeles *text1$ en tekst. Er resultatet uden for det tilladte område, sker en fejl, der opfanges af ON ERROR-sætningen, som hopper til rutinen *fehler. Resultatet for værdien 0, som også tildeles *erg%. I *ergebn- siger Amigaen resultatet af beregningen. I tilfælde af fejl siger den: (-den kan jeg ikke regne ud-).

4.2 Det hele går bedre med lidt musik

Normalt er det besværligt at programmere musik, men ikke med Amigaen. Man kan spille melodier, med SOUND-sætningen, uden besvær.

> **SOUND frekvens, varighed, lydstyrke, kanal**
> giver en tone af den nævnte frekvens, varighed og lydstyrke på den valgte kanal (der er fire til rådighed). SOUND starter blot lydfrembringelsen og vender tilbage til programmet. Tonen fortsætter efter programmet forlængst er igang med andre ting
>
> **SOUND WAIT**
> standser tonefrembringelsen midlertidigt
> **SOUND RESUME**
> igangsætter atter en tonefrembringelse, der var stoppet med SOUND WAIT

Det ser måske kompliceret ud, men begynd i det små og prøv en enkelt tone ved at indtaste følgende i direkte funktionsmåde:

```
SOUND 261.63, 18.2, 127
```

Det første tal giver frekvensen i Hertz (Hz) og kan ligge mellem 20 og 15000 Hz. 261.63 svarer til et C på musikens toneskala. Den anden talangivelse be-

stemmer tonens varighed (i taktslag, et sekund er 18.2 taktslag). Værdier mellem 0 og 77 taktslag er tilladt. Den sidste værdi repræsenterer lydstyrken og kan være mellem 0 og 255. Da der ikke er nogen kanalangivelse, gælder standardværdien 0, som betyder udlæsning via den venstre tonekanal.

Som man ser, er det ganske let. Med et ord og to til fire tal kan man frembringe de smukkeste toner.

REM Tonleiter
'P.4.2-1

Frekvenserne for de syv noder fra C til H lægges ind i feltvariablen f(0). De tages fra en DATA-sætning. I den anden række DATA-sætninger står henholdsvis varighed og frekvens af en tone hele tiden ved siden af hinanden. Programmet frembringer en stigende og faldende toneskala, der afspilles af rutinen song. Læg mærke til, at næste tone spilles, når den foregående er pænt afsluttet. I næste afsnit forbedrer vi skalaen væsentligt.

REM Tonleiter2
'P4.2-2

Den anden song-rutine kan spille tonerne C til H og c til h efter bogstaver, der læses fra DATA-sætningerne (og ikke efter frekvenser). Bogstaverne lægges fra starten i tekstvariablerne *note$ og de tilhørende frekvenser i feltvariablen f. Tonelængden behøver man ikke at angive i takter, men - som det er almindeligt i musik - i sekstendele af et sekund. Med dette lille program kan man spille en vilkårlig melodi, blot man har et partitur med noder og tonelængder. Til et længere musikstykke skal man forlænge DATA-sætningen og indlæsningsløkken (den øvre grænse for j). Prøv, hvordan det virker med en rigtig melodi.

REM Abendlied
'P4.2-3

Det lyder ganske godt, ikke? Frekvenserne står som enkelt-præcisions tal i listningen, og desuden er der dukket en node "P" op. Bogstavet P står for pause - vigtigt i alle musikstykker. P skrives altid i datasætningerne, når to ens toner følger efter hinanden, ellers ville de blive spillet ud i et. En lille pause imellem, får det til at lyde som et rigtigt instrument.

Til slut et eksempel, der er beregnet på to tonekanaler:

```
REM Volksweise
'P.4.2-4
```

De fleste vil sikkert synes, det lyder rædsomt. Eksemplet er blot til yderligere belysning af Sound-programmeringens teknik.

Programmet er lavet sådan, at visen spilles igen og igen i en løkke. Det sørger rutinen *ablauf for. Med RESTORE sætter man nodelæsningspointeren tilbage på det første element, og kalder så rutinen *song.

I song ser man de to SOUND-sætninger SOUND WAIT og SOUND RESUME. SOUND WAIT står i begyndelsen af WHILE-løkken. Sætningen standser afspilningen af nye toner. De følgende SOUND-sætninger lægger nye toner i en ventekø, indtil der fortsættes med SOUND RESUME. Dette trick bevirker, at alle kanaler starter på samme tid. Så kan tonerne fra køen afspilles.

DATA-elementerne til hver node indeholder tre værdier i dette program: fr$ for tonehøjde, variablen da til fastsættelse af tonelængde og endelig variablen laut til lydstyrken. Den anden stemme i programeksemplet er til rytmisk underlægning. Data-linierne er igen udformet overskueligt nok til, at man nemt kan indsætte egne noder. Bemærk, at ved den sidste node skal værdien nul indsættes for *da. Værdien sørger for, at *da bliver logisk falsk, så WHILE WEND-løkken kan forlades.

4.3 Toner - blide bølger eller kantede

Bestemme bølgens form.

> **WAVE kanal, definition**
> fastlægger bølgeformen for de toner, der lyder over en tonekanal

Tonekanalens nummer anvendes nøjagtig som i SOUND-sætningen. Tallene 0 og 3 gælder venstre og tallene 1 og 2 højre tonekanal. Til definition af bølgeformen anvendes nøgleordet SIN. Så frembringer tonekanalen sinusbølger (der lyder omtrent som et klaver). Det er den forudinstillede standardværdi og behøver ikke at angives, sålænge man kun arbejder med en tonekanal. Man kan også angive et numerisk felt med 256 elementer i stedet for SIN. Hvert element i feltet kan indeholde en heltalsværdi fra 127 til -128. Sådan kan man realisere en vilkårlig bølgeform og frembringe klangen af forskellige musikinstrumenter.

Lad os se på nogle forskellige bølgeformer. At det kun er en lille del af mulighederne, ses alene af, at der rent matematisk må være 256*256 muligheder. Den første bølgeform, vi vil lave om på, er sinusbølgen selv. Den kendes bl.a fra HiFi-reklamer.

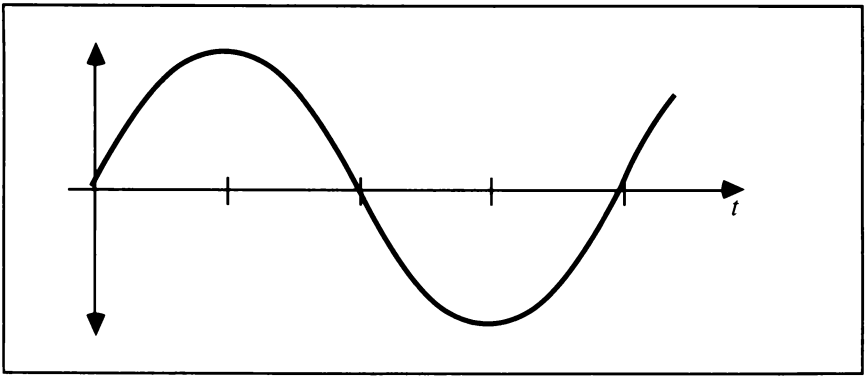


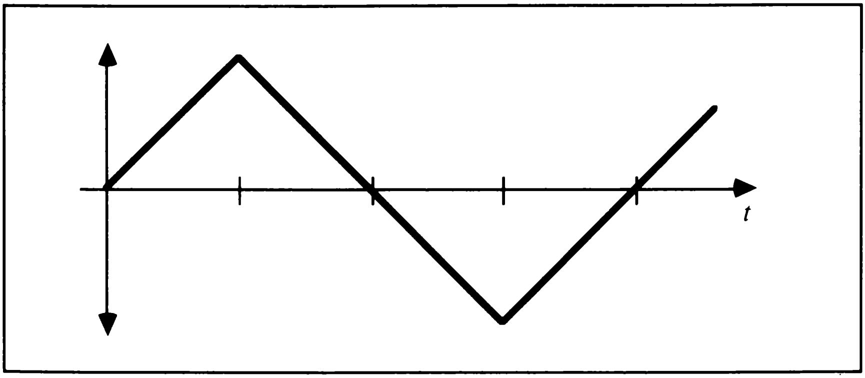
Fig 4.1: Sinusbølgen

REM Sinuswelle
'P.4.3-1

Fastlæggelsen af bølgeformen sker i rutinen *sinus. For at undgå besværet med at skulle beregne alle de 256 værdier, formen kræver i DATA-sætningerne, gør programmet det med Basic-funktionen SIN og lægger resultatet i feltvariablen *we(). (Bemærk at argumenterne til SIN-funktionen skal angives i radian og ikke i grader.) Rutinen afsluttes med WAVE-sætningen, der tildeler den i *we opbevarede bølgeform kanal nul.

I programdelen *ablauf får variablen *faktor forskellige værdier og bevirker, at sinusvingningen får varierende udslag. I den indre FOR NEXT-løkke spilles tre toner hver gang. Når programmet er spillet nogle gange, vil man bemærke, at tonen lyder tynd ved meget små udsving (små værdier af *faktor). Hvis svingningen forstørres, lyder den mere og mere fuldttonende. Klangen svarer ved fuldt udsving nærmest til et orgel, og ved mindre lyder det noget i retning af en fløjte. Det kommer også an på tonelængde og frekvens.

Den næste bølgeform er trekantbølgen. I modsætning til en sinussvingning stiger og falder den i lige linie.



Figur 4.2 Trækantsbølge

REM Dreieckswelle
'P.4.3-2

Programopbygningen er den samme som i første eksempel. Bølgeformen fastlægges i rutinen *drei. Da bølgen stiger kontinuerligt op og ned, kræves ingen indviklede beregninger som med sinusbølgen. Der spilles fire toner med en længde på 12 takter. Sammenlignet med sinussvingningen lyder tonen nu mere hult, omtrent som en trompet. Prøv de forskellige former.

Næste bølge har også en trekantet form; den stiger og falder ikke med samme hastighed, men det går gradvis opad og stejlt (næsten lodret) ned. Man kalder det en savtandsbølge.

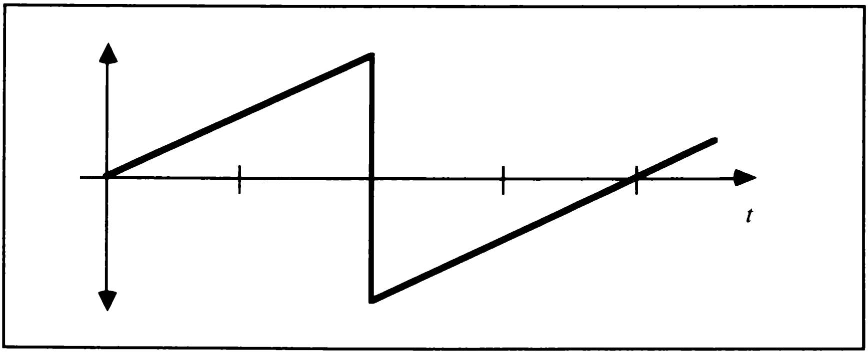


Fig 4.3 Savtakbølgen

REM Saegezahnwelle
'P.4.3-3

Når man sammenligner savtandsbølgens toner med trekantbølgens, synes sidstnævnte at være mere rungende. Prøv savtandsbølgen i hele frekvensområdet med forskellig bølgelængde, så får man et bedre indtryk af de forskellige anvendelsesmuligheder.

Endnu en form er pulsbølgen, også kaldet den variable pulsbølge. Den svinger som en firkant. Bredden af firkanten, pulsbredden kan man selv bestemme.

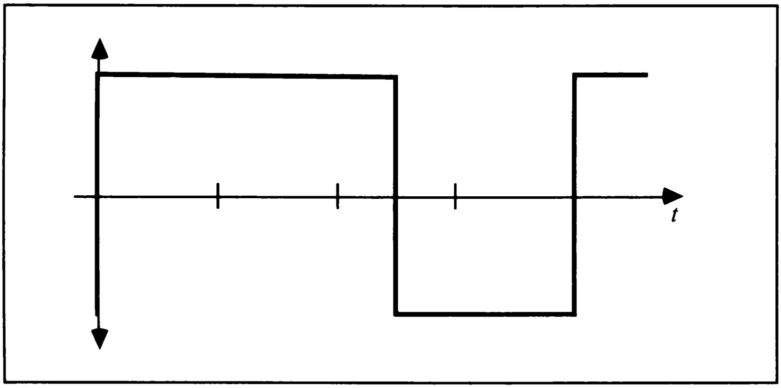


Fig 4.4: En pulsbølge

REM pulsbølge
'P.4.3-4

I programdelen *ablauf gives variablen faktor forskellige værdier, hvorved der fastlægges forskellige pulsbredder. Før de fire testtoner spilles, hoppes der til rutinen *puls, som fastlægger bølgeformen (afhængigt af faktor). Som det ses, forekommer der kun maksimale (+127) og minimale (-127) svingningsværdier. Alt efter pulsbredde giver det forskellige klangfarver, der undertiden lyder som et spinet, men ofte kun som støj.

Den sidste bølgeform, vi skal se på, kaldes "hvid støj". I modsætning til andre bølger, er bølgeformen (værdierne i bølgeform-feltets felter) tilfældige. (RND-funktionen, som bruges hertil, er forklaret i kapitel 5.4.

'P.4.3-5

De toner, der frembringes af denne bølgeform, lyder meget forvrænget. Man kan få elektronisk klingende lyde og opnå specielle effekter. Man kan igen ændre svingningernes bredde med *faktor.

Nu har vi lært de vigtigste bølgeformer at kende, og bør huske, at det kun er en lille del af det vældige spektrum af muligheder.

Til afslutning skal vi spille en sang. For at få noget ud af programmeringen, er der anvendt en ny bølgeform for hver af de fire kanaler. Det er der kommet en gammel vise ud af, som slet ikke lyder gammel:

REM schwabische Weise
'P.4.3-6

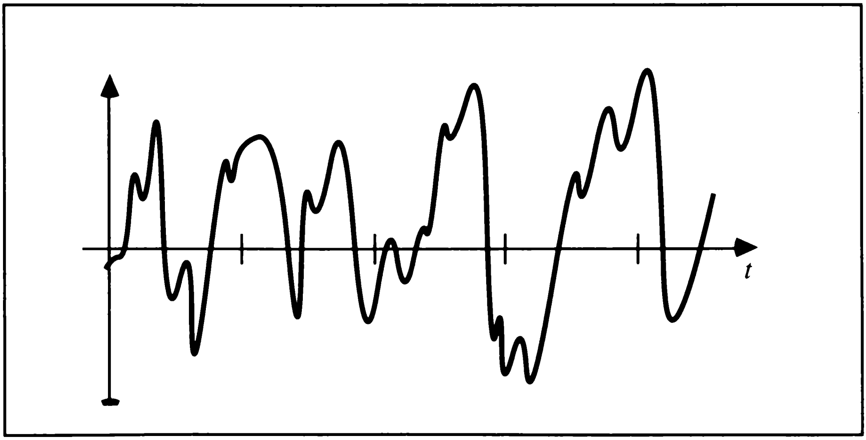


Fig 4.5 Hvis støj

Jeg har forsøgt at få melodiske toner ud af de forskellige bølgeformer. Om deter lykkedes, må læseren afgøre. Med lidt musikforstand kan det nok gøres bedre. I næste kapitel skal vi derimod forsøge at få lidet melodiske toner ud af Amigaen.

4.4 Crash, zoom & wham - lydeffekter

Måske har læseren prøvet at lede efter en bestemt lyd til et hjemmelavet program. Måske ved programmering af spil. Der præsenteres en række lydeffekter, man kan indbygge i egne programmer; man skal ikke regne med en

passende lyd til alle lejligheder. Har man ikke den rigtige lydeffekt, går det måske ved at lave lidt om på eksemplerne.

Når man har fremstillet en bestemt lyd, sker det alligevel, at den ikke lyder godt i programmet. Det er en helt anden sag at lade en lyd køre alene, end i et program. Sålænge man lader en række lyde afvikles separat i et underprogram, og ikke påvirker programforløbet ved interrupt-aflæsninger, kan der ikke ske ret meget; men mange lyde skal fremkomme parallelt med en begivenhed. Hvis man f.eks. affyrer et laserskud i et spil, skal tonen være til stede, sålænge man kan se sporet af skuddet. Så lyden skal tilpasses til programmets timing. Måske kan man nøjes med at rette på værdien af tonelængden i SOUND-kommandoen; i modsat fald er det et større arbejde.
Bølgeformen til programmering af lyde er den i begyndelsen hæslige hvide støj; men man kan også opnå nogle flotte effekter med pulsbølgen. I det første eksempel anvendes disse to bølgeformer til et bilhorn, der går gennem marv og ben.

REM hupe
'P.4.4-1

Når man spiller tonen et par gange, bemærker man, at den lyder lidt forskelligt hver gang, men alligevl bliver ved med at lyde som et bilhorn. Årsagen er bølgeformen *rau. Programmet får fire gange under fastlæggelsen af bølgeformen funktionen RND til at begynde med nye værdier. Det er derfor fuldstændigt umuligt først at skabe bølgeformen, når der skal bruget i programmet, det ville tage alt for lang tid med beregningerne. Bølgeformen skal være beregnet på forhånd og gemt. Det bedste tidspunkt til det er altid programmets start; derfor er programrutinerne opdelt i to afsnit. Det første afsnit *programmanfang indbygger vi så tidligt som muligt i programmet. Den anden rutine, der lader lyden fremkomme, kan så anbringes et vilkårligt sted. Og glem ikke at forsyne feltet til bølgeformen med et %-tegn.

Nu til det andet eksempel, alarmtrin1

REM Alarmstufe1
'P.4.4-2

Man hører den dybe vedvarende brummen af et signalhorn. Denne lyd kaldes også "red alert". Den kan f.eks. anvendes til at sætte et rumskib i alarmberedskab, eller som alarmsignal ved et bankrøveri. I dette lille program bruges en pointer til programstyring. Hvis rutinen *programmanfang først er kaldt, kører den, til den afbrydes af en anden begivendhed (ON TIMER, ON

COLLISION). Den standser først, når *zeiger er ulig 9. Så længe programmet står alene, kan det kun afbrydes af menukommandoen Stop.

Straks til det tredje eksempel: brandvæsenet kommer!

REM brandvæsen
'P.4.4-3

Programmet overrasker med en amerikansk brandbils nervesønderrivende hylen. Tonefrembringelsen beror på den trinvise (STEP 10) ændring af frekvensen op og ned mellem 700 og 1000. Den hvide støj frembringes modsat bilhorneksemplet i blot to løkker. Desuden begrænses udslaget i svingningerne til en bredde på kun 20.

Det næste program anvender også pulsbølgen.

REM Masch.Gewehr
'P.4.4-4

(rutinen puls er ikke længere trykt i listningen, tag den fra programmet *Alarmstufe1). Når programmet kører, lyder den korte, tørre knitren af en maskingeværsalve. Den korte, hule klang opnås ved at nedsætte pulsbredden til 1/254 af en bølges varighed.

I det følgende program anvendes pulsbredden atter "normalt".

REM Abklang
'P.4.4-5

Heller ikke her er programdelen puls listet. (Hvis man i de følgende programmer i dette kapitel kun finder hopmærket, men ikke rutinen til en bølgeform, findes den i et andet program og skal kopieres derfra.) Dette programs tonerække lyder ganske effektfuldt, og kan bruges til forskellige formål såsom nedskydningen af et eller andet objekt i et skydespil. Prøv en gang, hvordan det lyder, hvis man ændrer den første linie før *Abklang som følger:

```
FOR n=5 TO 40
```

Heller ikke så dårligt, vel?

Også det næste program ændrer frekvens og tonevarighed i en bestemt rækkefølge:

REM Ufo-landet
'P.4.4-6

Programmet frembringer en imponerende række lyde, der ville passe godt til en landende UFO, eller affyringen af en raket. Nu har vi tre lydprogrammer efter hinanden med pulsbølger. Kig en gang på variablerne b1 og b2, der bestemmer udseendet af de tre pulsbølger. Man ser, hvilken forbavsende virkning, der opnås ved at variere pulsbredden.

I det følgende program lyder derimod atter den hvide støjs bølgeform.

Dette program lyder som en smukt snurrende motor. Det viser, hvor lille et kerneprogram indenfor WHILE-løkken, man behøver for at få en effektfuld lyd. Prøv at sammenligne de enkelte værdier med dem i programmet *hupe. Man vil se, at bortset fra frekvens og tonelængde er alle parametre de samme. En god tilskyndelse til egne eksperimenter, ikke?

Det næste program anvender igen den simpleste bølgeform, den hvide støj.

REM Hochspannung
'P.4.4-8

Når programmet startes, lyder den dybe brummen af en højspændingstransformator. Dette miniprogram er også et godt eksempel på, at man kan frembringe forbavsende lyde med lidet besvær.

De hidtidige eksempler har begrænset sig til en enkelt tonekanal; men man kan også fastsætte indtil fire bølgeformer (på fire kanaler) til sin lydrutine. Så skal man huske at tildele hver bølgeform sin egen feltvariabel og naturligvis først definere den og fylde den med værdier.

Det sidste eksempel arbejder på den beskrevne måde med to bølgeformer på én gang.

REM Laser
'P.4.4-9

Programmet frembringer den typiske lyd af et laserskud i et action-spil. Fordi der anvendes to forskellige pulsbølger, er alle sætningerne i *programmanfang dobbelt til stede.

Det var så kapitlet om lyde. Nu skulle læseren have et lille arkiv med lydeffekter, som kan udbygges og forbedres efter behov. I øvrigt kan jeg kun opfor-

dre til, at man selv eksperimenterer videre med varigheder, frekvenser og bølgeformer på en eller flere kanaler med forskellige parametre.

4.5 Levende billeder - sprites og BOBs

En særligt spændende form for computergrafik er arbejdet med levende billeder (fagudtrykket er "animation"). Hidtil har vi kun anvendt PUT-sætnngen til det, nu skal vi lære et stort antal nye kommandoer at kende, der næppe lader noget tilbage at ønske. Alligevel er det let at håndtere de fleste af disse sætninger.

Amiga-Basic har to typer bevægelige objekter

- Sprites, der bevæges af speciel hardware
- BOBs, der (hovedsagelig) bevæges af software

I dette afsnit vil vi først beskæftige os med Sprites. De data, der bestemmer en sprites' udseende, lægges i en strengvariabel og kan også anbringes i en sekventiel fil. Dataene indeholder alle oplysninger om objektet såsom længde, bredde, form osv. (men ikke sprite'ns position). Man skal altså ikke spekulere over, hvordan man får disse oplysninger ind i en fil. Arbejdet overtages for størstedelen af en "Objekt-Editor", et program, der ligger i skuffen "BASIC-Demos"

Der er fire farver til rådighed til farvning af sprites. Den først farve virker ganske vist senere gennemsigtig, så der i virkeligheden kun er tre effektive farver; men de pågældende farver behøver ikke at være identisk med nogen fra den aktuelle billedskærms palet. Man kan også repræsentere mere end fire farver i Workbench'ens firfarve-mode. Antallet af sprites, der kan vises, er begrænset til otte.

Luk nu skuffen "BASICDemos" op og start programmet "ObjEdit" med et dobbeltklik på dets ikon. Efter en kort pause skriver programmet:

```
Enter 1 if you want to edit sprites
Enter 0 if you want to edit bobs
```

Tryk 1 (sprites) og så RETURN. Tegneområdet kan kun indstilles i højden (y), da bredden af sprites er fast indstillet på 16 pixels (y = 15). Vælg straks forstørrelsen 4x4 fra Enlarge-menuen, ellers er det svært at se noget. De øvrige værktøjer i Tools-menuen kan man godt glemme, de er ikke til at bruge med

den lille bredde. Tegn nu en sprite; et eller andet tilfældigt. Vælg derefter Save As fra Tools-menuen og gem sprite'n under navnet "test".

Forlad så programmet og luk skuffen BASICDemos. Luk den op igen og tag ikonen "test" ud. Læg den i en skuffe, der benyttes til at gemme egne programmer (eller i en ny skuffe). Indtast det følgende program og gem det i den samme skuffe under navnet "sprite-test".

REM Sprite-Test
REM P.4.5-1

Start nu programmet (efter at have gemt det). Som det ses, er et sådant miniprogram nok til at bringe den første sprite på skærmen. PALETTE-sætningen i den anden linie sørger for, at sprite'n bliver tydeligere synlig, en sort sprite er lettere at se. Kig lidt nærmere på programmet. Efter åbningen af filen "test", som indeholder sprite-dataene, finder vi følgende nye sætning:

> **OBJECT.SHAPE objektnummer, streng**
> frembringer et nyt objekt (sprite eller BOB) med betegnelsen objektnummer og tager den nødvendige information fra den overgivne streng
>
> **OBEJCT.SHAPE objektnummer1, objektnummer2**
> frembringer et nyt objekt (sprite eller BOB) med betegnelsen objektnummer1 og tager den nødvendige information fra det alleredee ksisterende objekt med betegnelsen objektnummer2.

Med OBJECT.SHAPE-sætningen overgives først alle sprite'ns data, såsom størrelse, form osv til programmet. Dernæst fastlægges det med to sætninger, hvor på skærmen sprite'n skal fremkomme.

> **OBJECT.X objektnummer,X**
> **OBJECT.Y objektnummer,Y**
> bevæger et objekt (sprite eller BOB) med betegnelsen objektnummer, såat dets øverste venstre hjørne får x-koordinaten X, resp. y- koordinaten Y (uden at den anden koordinat ændres).

De nævnte værdier for x- og y-koordinater refererer som sædvanligt til det øverste venstre hjørne af skærmen. Disse sætninger bevæger godt nok sprite'n, men gør den ikke synlig. Det tjener den sidste nye kommando i dette program, OBJECT.ON, til.

OBJECT.ON objektnummer1,objektnummer2
OBJECT.OFF objektnummer1, objektnummer2
gør objekterne (sprites eller BOBS) med de nævnte betegnelser synlige (OBJECT.ON) eller får dem til at forsvinde (OBJECT.OFF)

I begyndelsen af afsnittet talt vi om egentlig bevægede objekter. Men her står sprite'n stille på skærmen. Det skal vi straks ændre ved at tilføje følgende linier til programmet "sprite-test".

```
OBJECT,VX 1,-10
OBJECT.VY 1,-1
OBJECT.START
vaerdi=1
WHILE vaerdi
SLEEP
WEND
```

Når man nu starter programmet, bevæger sprite'n sig mageligt fra højre til venstre side af skærmen og klatrer endda et lille stykke op ad den. Det skyldes tre sætninger, der giver sprite'n en hastighed og sætter bevægelsen i gang.

OBJECT.VX objektnummer, hastighed
OBJECT.VY objektnummer, hastighed
fastlægger hastigheden i X- eller Y-retningen (i pixels pr. sekund) for et objekt (sprite eller BOB) med betegnelsen objektnummer.

På x-aksen betyder en positiv værdi af hastighed en bevægelse mod højre og en negativ en bevægelse mod venstre. På y-aksen giver et negativt tal en bevægelse opad, og en positiv værdi bevæger sprite'n nedad. Det er interessant, at bevægelsen sker helt uafhængigt af Basic-programmet og vedbliver, til sprite'n støder ind i noget. Man behøver derfor ikke at programmere kunstfærdige løkker for hele tiden at bevæge objektet et stykke videre. Det normale programforløb forbliver helt uberørt heraf. Men man skal ganske vist give "startskuddet" før et objekt begynder at bevæge sig.

OBJECT.START objektnummer1,objektnummer2,...2
OBJECT.STOP objektnummer1,objektnummer2,...
sætter alle objekter (hvis der ikke er angivet nogen betegnelser), eller kun dem med de nævnte betegnelser, i bevægelse, resp. stopper bevægelsen (OBJECT.STOP)

Det var en hel mængde nye sætninger. Lad os straks se, hvordan deres anvendelse med flere forskellige sprites kommer til at se ud.

REM Bummelbahn
REM P.4.5-2

Til dette program, "*Bummelbahn" er det ikke nødvendigt at konstruere nogen sprites. Data hertil findes nemlig i programmet selv (i DATA-sætningerne). Ved den første start af programmet skrives disse data i tre filer, og indlæses så ved senere programkørsler derfra. Ved starten forsøger programmet først at åbne denne fil. Lykkes det ikke, gås der (via ON ERROR-sætningen) til rutinen *fehler. Her læses sprite-værdierne fra DATA-sætningerne og gemmes så under filnavnene "*lok1", "*lok2" og "*wag. Så går det retur til *anfang, hvor filerne nu bliver fundet.

Efter at de tre sprites (to lokomotiver og en vogn) er indlæst fra filerne, dannes sprites 4 og 5 ved kopiering af den tredje sprite (vogn). Så tegnes der en jerbanedæmning med spor på skærmen. Med sætningerne OBJECT.X og OBJECT.Y sættes til sidst hele toget på sporet og gøres synligt med OBJECT.ON. Bevægelsen af bumletoget og kontrollen dermed sker i den følgende WHILE-løkke.

I WHILE-løkken afprøver den første IF-sætning, om lok har nået den venstre skærmkant. I så tilfælde bliver alle sprite'nes hastighed OBJECT.VX positiv, og toget bevæger sig mod højre. Den anden IF-sætning har en tilsvarende funktion, denne gang for højre billedkant; hastigheden OBJECT.VX af alle sprite'ne bliver negativ og toget kører videre fremad mod venstre. Det interessanteste ved dette program ligger i løkkens to sidste IF-sætninger. Ved hvert løkkegennemløb forhøjes variablen a med én. Så snart HILE-løkken er gennemløbet tyve gange, gøres det først lok usynligt med OBJECT.OFF og det andet lokomotiv synligt med OBJECT.ON. Først fastslås positionen af det ene lok med OBJECT.X-funktionen og overgives til det andet lok.

> **v=OBJECT.X(objektnummer)**
> **v=OBJECT.Y(objektnummer)**
> giver den aktuelle X-, resp. Y-position af objektet (sprite eller BOB) med betegnelsen objektnummer

Skiftet mellem de to damplokomotiver går så hurtigt, at man knapt bemærker det på skærmen. Da de to loks imidlertid indeholder forskelligt anbragte røgskyer, ser det ud, som om røgskyerne bevæger sig.

Det var det vigtigste om, hvad man kan gøre ved sprites med Basic. Vi vender os nu til BOBs, men læseren har sikkert fået lyst til andre eksperimenter med sprites.

4.6 Kunstflyvning

BOB er en forkortelse for Blitter OBject. Blitteren er en chip inden i Amigaen, der bl.a. sammen med speciel software har til opgave at flytte billeder hen over skærmen.

BOBs kan konstrueres i næsten enhver størrelse. Der er ingen grænser hverken for deres antal eller for antallet af farver, de indeholder. (BOBs kan indeholde så mange eller så få farver, som den skærm de vises på.) Derimod kan de ikke bringe ekstra farver på skærmen sådan som sprites. Det er de vigtigste forskelle på BOBs og sprites. Hvis man arbejder med "ObjEdit", skal man passe nøje på farvernes rækkefølge, da dette program i reglen arbejder med andre farver, end der senere bruges i programmet.

Programmet "ObjEdit" er fra starten af indstillet til fire farver, dvs. to bitplaner; men det lader sig relativt let omstille til tre bitplaner, altså otte farver. Vi skal også udnytte denne mulighed i et senere program. Prøv dog først med en firfarvet BOB. Gå frem nøjagtig på samme måde som beskrevet for sprites i sidste afsnit. Start "ObjEdit" og indtast efter spørgsmålet om objekttype et nul for BOBs. Gem BOB'en under navnet "bob-test". Derefter skal der laves om på en linie i programmet "sprite-test" (se ovenfor). Start det ændrede program og betragt resultatet.

fra: OPEN "test" FOR INPUT AS 1

til: OPEN "bob-test" FOR INPUT AS 1

Er læseren en smule utilfreds med sin første BOB? Her har vi en supercomputer med alle mulige farver, og så må vi nøjes med blot fire farver. Det lyder heller ikke så imponerende, at objekt-editoren kan vise otte farver. De følgende sætninger viser derfor, hvordan man får den til at frembringe BOBs med 32 farver.
Programmet "ObjEdit", må ændres lidt. Når programmet spørger om arten af objekter, sprites eller BOBs, så afbryd med Stop fra Run-menuen. Lad listningen vise med Show List fra menuen Windows. Lav vinduet størst muligt, og bladr med tastkombinationen SHIFT-CURSOR DOWN (pil nedad) til den anden skærmside. Her skal første ændring foretages

Lav de følgende linier om

```
WinY =185: WinX=617
'If BOBs are to be created with other than 2-bit planes
' alter next 3 lines (only if machine has more than 256k)
' Depth=3
' scrn=1
' SCREEN scrn,640,200,Depth,2
' WINDOW 1,,(0,0)-(WinX,WinY),31,scrn
```

til:

```
WinY=170: WinX=270
' If BOBS are to be created with other than 2 bit-planes
' alter next 3 lines (only if machine has more than 256k)
Depth=5
scrn=1
SCREEN scrn,320,200,Depth,1
WINDOW 1,,(0,0)-(WinX,WinY),31,scrn
```

Den anden ændring vedrører de to rutiner PrintColorBar og CheckColor som ligger efter hinanden tæt ved listningens afslutning. Lav følgende linier om (programlinier, der ikke behøver at ændres, er kun antydet):

```
PrintColorBar:
COLOR...
LOCATE...
ColorBar...
COLOR...
x=70
FOR i=0 TO maxColor
LINE (x,ColorBar)-(x+20,y+ColorBar+10),i,bf
LINE (x,ColorBar)-(x+20,y+ColorBar+10),1,b
x=x+20
NEXT i
RETURN

CheckColor:
IF...
IF...
i=INT((CurrentX-70)/20)
...
```

til:

```
PrintColorBar:
COLOR...
LOCATE...
ColorBar...
COLOR...
x=70
FOR i=0 to maxColor
LINE (x,ColorBar)-(x+6,y+ColorBar+10),i,bf
LINE (x,ColorBar)-(x+6,y+ColorBar+10),1,b
x=x+6
NEXT i
RETURN

CheckColor:
IF...
IF...
i=INT((CurrentX-70)/6)
...
```

For ikke at skulle ændre programmet, hver gang man vil fremstille 32-farvede BOBs, er det bedst at gemme det under navnet "ObjEdit32". For at se farverne, konstruerer vi to nye BOBs. Start det nye program "ObjEdit32". Konstruktionsvinduet fremkommer med det samme. Fordi programmet er ændret mindst muligt, skal man passe på følgende:

NB! *Vælg ALDRIG forstørrelsen ENLARGE 4x4!*

Det vil ødelægge alt, hvad der er tegnet indtil nu. Man kan tegne i det forstørrede felt, men ikke gemme noget. Der kan arbejdes med alle værktøjer og 32 farver i konstruktionsvinduet. Man kan forstørre eller formindske selve vinduet. Men bemærk, at farverne i de 32 kasser ikke svarer til de farver, der bagefter skal anvendes i programmet.

Tegn to BOBs. Den første med målene ca. x=30 og y=28. Navnet skal være "bob32". Lad den anden være en ordentlig bamse, med målene x=93 og y=84. Døb den "test32". For at få indtryk af farverigdommen, skal man anvende så mange farver som muligt. Gå ud af programmet, luk Demo-skuffen og åbn den igen. Træk så de to ikoner med BOB-data ud og læg dem sammen med følgende program i en anden skuffe.

REM Farbenpracht
REM P.4.6-1 BOBs in 32 Farben

Sandsynligvis virker programmet nu lidt for broget; men det er kun for at demonstrere alle farvemulighederne. Man kan vælge mere smagfuldt senere. Med SCREEN- og WINDOW-sætningerne blev der fastsat en skærm på 320 pixels bredde og 200 pixels højde. PALETTE-sætningen definerer så de 32 mulige farver heri. I følgende FOR-løkke fyldes skærmen med 32 forskelligt farvede rektangler. Så indlæses de to BOBs og anbringes med de bekendte OBJECT-sætninger. Den følgende WHILE-løkke sørger for, at de to BOBs bevæger sig langsomt fra side til side. Desuden hoppes ved hver løkkegennemgang til rutinen *tast, der giver mulighed for at slutte programmet. Program viser tydeligt mulighederne med BOBs; men det har en mangel, som egentlig ikke gør ret meget. Derfor et lille spil, hvor otte-farvede BOBs bevæger sig.

Programmet "*Kunstflug" er en test af reaktionsevne. Spilleren skal flyve i lav højde frem og tilbage mellem en række træer. Træernes position ændres og maskinen flyver hurtigere for hver omgang. Jo længere spilleren kan holde flyet i luften, des flere points får han. Fordi træerne er ret store BOBs, har det ikke været muligt at undgå en vis flimren - men det generer ikke meget.

For at programmet ikke skal fylde for meget, har jeg undgået programmerede sprites. Man skal konstruere to BOBs med Object-Editoren. Vi sætter ObjEdit på tre farveplaner, dvs. 8 farver. Fjern de fire apostroffer i begyndelsen af programmet, som vi tidligere gjorde det. Man behøver ikke flere ændringer. De enkelte farver har følgende værdier:

PALETTTE-sætningens farvenummer:	Farvetone
0 til 2	lyseblå
3	lyserød
4	sort
5	mørkegrøn
6	mellemgrøn
7	lysegrøn

Tabel 4.1: Farvepaletten i programmet *Kunstflug

Den første BOB "*baum-" skal forestille et træ set oppefra. Det er bedst at anvende farverne 5 og 7. Størrelsen skal være ca. 83 (X) gange 40 (Y). Den anden BOB "*flugz bliver til en flyvemaskine med målene X=61 og Y=24. Flyet skal tegnes med propellen mod venstre. Rødt og sort anbefales. Læg de to BOBs sammen med det følgende program i en anden skuffe end "BASICDemos".

REM Kunstflug
REM P.4.6-2

Bemærk, at det kræver et joystick at kunne spille dette program. Har man ikke det, må man stille sig tilfreds med følgende forklaringer: Lige i begyndelsen finder man en ny sætning ON COLLISION. Den tjener som ON MENU eller ON MOUSE til at dirigere programmet, når der indtræder bestemte hændelser på bestemte steder.

> **ON COLISION GOSUB hopmærke**
> sørger for at subrutinen ved hopmærke kaldes, når et objekt (sprite eller BOB) støder sammen med et andet objekt eller med billedranden

Dermed kan man angive, hvad der skal ske, når eet objekt støder sammen med et andet. For ikke at få yderligere afbrydelse ved en kollision (handlingen går jo videre), udføres samtidig COLLISION STOP (se nedenfor). I programmet forgrenes ved hver kollision til *zusammen. Her spørges blandt andet, om man vil spille videre. Hvis han trykker j, sættes værdien af *zeiger på nul og forlader dermed automatisk hovedprogrammets WHILE-løkke. Et direkte hop til start er umuligt, da det ville bringe vild forvirring i Amiga-Basic. Der skal nemlig altid følge et RETURN efter et GOSUB (og et NEXT efter FOR, ligesom et WEND efter WHILE). Man skal derfor undgå at hoppe direkte ud af subrutiner eller løkker og i stedet gøre som her. Men tilbage til programmet. ON COLLISION-sætningen skal, som enhver anden ON-sætning, aktiveres af COLLISION ON, før der sker noget.

> **COLLISION ON**
> sætter en forudgående ON COLLISION-sætning i kraft

> **COLLISION OFF**
> ophæver virkningen af den sidste ON COLLISION-sætning

> **COLLISION STOP**
> ophæver forbigående virkningen af den netop gældende ON COLLISION-sætning, indtil der igen udføres et ON COLLISION (Amiga-Basic noterer sig herunder alle kollisioner i en art kø og udfører dem efter hinanden efter COLLISON ON)

Efter programstyringskommandoerne åbnes et vindue i en screen med otte farver, som fastlægges med PALETTE. De første tre farver er ens. Det bevirker, at vinduesrammen forsvinder helt. Så defineres tre træer og et fly med OBJECT.SHAPE. Ved start begynder det egentlige programforløb med fast-

sættelsen af ekstra egenskaber af BOBs'ene, og anbringelsen af fly-BOB'en. I den påfølgende WHILE-løkke udlæses det aktuelle antal points; ved overskridelse af en bestemt x-position med flyet startes en ny omgang, og joysticket aflæses.

I *unterbr fastlægges for hver BOB, hvilket sammenstød med et andet objekt, der fremkalder en afbrydelse.

OBJECT.HIT objektnummer,egenværdi,fremmedværdi
fastlægger, med hvilke andre objekter et objekt med betegnelsen objektnummer skal støde sammen for at fremkalde en afbrydelse.

Normalt fører ethvert sammenstød med et andet objekt eller med vindueskanten til en afbrydelse (og en forgrening med ON COLLISION). OBJECT.HIT tillader en begrænsning af antallet af farlige objekter. For at bestemme værdier af egenværdi og fremmedværdi, må vi atter gå til laveste niveau i Amigaen, bit-niveauet. Der er 16 bits for hver af de to værdier. Ved en kollision sammenknyttes de to værdier logisk med AND. Kun hvis begge værdier er sat (lig med 1), sker der en afbrydelse. Den sidste (nulte) bit er reserveret til sammenstød med vindueskanten. Hvis den ikke er sat, sker der ikke noget ved sammenstød med kanten. Lad os prøve:

	Egenværdi	Fremmedværdi
OBJECT.HIT 1,2,28 (fly)	00000000 00000010	00000000 00011100
OBJECT.HIT 2,4,2 (træ)	00000000 00000100	00000000 00000010
OBJECT.HIT 3,8,2 (træ)	00000000 00001000	00000000 00000010
OBJECT.HIT 4,16,2 (træ)	00000000 00010000	00000000 00000010

Tabel 4.2: Anvendelsen af OBJECT.HIT

Med egenværdi bag OBJECT.HIT defineres en bit for hvert objekt, der identificerer objektet ved sammenstød. I ovenstående tabel får flyet den anden bit fra højre (2), det første træ den tredje bit (4), det andet træ den fjerde (8) og det tredje træ den femte (16). Bit'en længst til højre er altid reserveret til billedkanten. Der må altid kun være en værdi sat (lig 1) i egenværdien. Eftersom egenværdien omfatter 16 bits, kan der maksimalt identificeres 15 objekter (plus skærmbilledkanten).

Fremmedværdien angiver, hvilke andre objekter et objekt (hvis betegnelse følger efter OBJECT.HIT) skal støde sammen med for at fremkalde en afbrydelse. Hvis et objekt støder sammen med et andet, så sammenknyttes det andets egenværdi med det førstes fremmedværdi. En afbrydelse (og en for-

grening via ON COLLISION) finder kun sted, når AND-sammenknytningen af disse tal giver et tal, hvori der mindst er sat en bit (der altså er ulig nul).

I næste rutine *vorfahrt forekommer en ny kommando. Den er ikke så kompliceret som OBJECT.HIT-kommandoen.

> **OBJECT.PRIORITY objektnummer, prioritet**
> fastlægger, at objekter med højere prioriteter dækker objekter med lavere prioriteter ved overlapning.

Ved at skubbe grafiske objekter oven på hinanden kan man skabe en illusion af rumlighed. Hvis prioriteten ikke fastlægges, får alle obejkter samme prioritet og dækker hinanden tilfældigt. Prioritetsværdien kan ligge melllem -32768 og +32767. I programmet "*Kunstflug" har alle tre træer højere prioritet end flyet. Det bevirker, at flyet flyver bag træerne (alligevel sker der en kollision).

I sidste rutine, *baumpo forhøjes flyets hastighed i trin på fem til maksimalt 60 pixels i sekundet. For at spilleren kan have udbytte af den større hastighed, forhøjes hans pointantal for hver omgang, han gennemfører uden uheld. Anbringelsen af træerne overlades til tilfældigheds-generatoren. Så er man sikker på, at man ikke på forhånd kan regne træernes placering ud.

4.6.1 Joystick'et

Joystick'et er en styrepind, hvormed man - som med musen - kan overføre bevægelser til skærmen og udløse events. Hvilket joystick man anvender, afhænger af, hvad man bruger det til. Det skal passe i stikket; men det skulle ikke volde problemer. Man kan tilkoble to joysticks til Amigaen. Stikket er anbragt lige ved siden af (eller bagved) musestikket. Det andet joystick sættes i musestikket. Vil man ikke give afkald på musen, kan kun bruges et joystick.

For at kunne omsætte joystick'ets bevægelser til skærmen, skal man vide, hvilke bevægelser joystick'et aktuelt udfører. Det har Amiga-Basic en funktion parat til.

> **v=STICK(n)**
> giver tilstanden af joystick'et i port 1 eller port 2 som et tal mellem -1 og 1

Funktionen STICK giver alt efter n følgende værdier:

n	Joystick og retning	Værdier og bevægelser
0	stick 1(A) i x-retningen	1 = til højre 0 = ingen -l = til venstre
1	stick 1(A) i y-retningen	1 = opad 0 = ingen -1 = nedad
2	stick 2(B) i x-retningen	1 = til højre 0 = ingen -1 = til venstre
3	stick 2(B) i y-retningen	1 = opad 0 = ingen -1 = nedad

Tabel 4.3: Resultater af STICK-funktionen

Hvorfor blev der anvendt tal og bogstaver til at skelne mellem de to joystick. Forklaringen ses i Basic-manualen. Her finder man bogstaverne A og B i joystick-forklaringerne. For at gøre det mere kompliceret står der 1 og 2 på de tilsvarende stik på Amigaen. I det følgende vil vi kun betegne joystick'ene efter stikkene (portene), altså som 1 og 2.

Betragt tabellen med de mulige resultater af STICK. Hvordan kan man bedst fastslå den retning, brugeren netop trykker styrepinden i?

STICK-funktionen giver de værdier, vi behøver til positionsændring! Den følgende Basic-linie giver nøjagtigt det vi ønsker.

x=x+STICK(2)

Dermed har vi undersøgt og videregivet to retninger. Hvis man ønsker en hurtigere bevægelse, ganger man blot resultatet af funktionen med en bestemt faktor:

x=x+2*STICK(2)

Bevægelsesændringen er fordoblet. Her et lille program, der lader spilleren behandle joystick'et efter forgodtbefindende:

REM Steuerknueppel
REM P.4.6.1-1 STICK STRIG

Som bevæget objekt anvendes BOB'en "*flugz" fra programmet "*Kunstflug". I programdelen *joyabfr findes værdierne for joystick'ets x- og y-bevægelser, og de videregives til BOB'en. Derefter undersøges joystick'ets affyringsknap med STRIG.

v = STRIG(n)
overgiver affyringsknappens status til joystick 1 eller 2

Alt efter n og tilstanden af den pågældende affyringsknap giver STRIG så følgende værdier:

n	STRIG(n)	Betydning
0	1/0	knappen på stick 1 blev nedtrykket/ikke nedtrykket
1	1/0	knappen på stick 1 er nu nedtrykket/ikke nedtrykket
2	1/0	knappen på stick 2 blev nedtrykket/ikke nedtrykket
3	1/0	knappen på stick 2 er nu nedtrykket/ikke nedtrykket

Tabel 4.4: Mulige resultater af STRIG

I programmet spørges med STRIG(3), om joystick 2 (ved bageste stik) er nedtrykket nu. Den anden undersøgelsesmulighed, STRIG(2), ville også give 1, hvis joystick'et var blevet nedtrykket siden sidste STRIG-kald, men atter var blevet sluppet. (STRIG(3) giver kun 1, når knappen stadig holdes nede.)

Hvis man trykker på affyringsknappen, fremkommer en hvid linie (skud?) foran flyet; den forsvinder igen, såsnart man slipper knappen. Flyv et par kurver med nedtrykket affyringsknap, så skrives der smukt på himlen. Vinduet kan forlades foroven, forneden og til højre; men hvis man forlader det til venstre, gennemløbes den indre WHILE-løkke først og så afsluttes spillet.

4.7 Nedskydning

Efter det vi i de to sidste afsnit har lært om sprites og BOBs, vil vi lige resumere forskellen på dem i en tabel.

Sprites	BOBs
fremkaldes alene af hardware	styres af software
stor hastighed	langsommere end sprites
maksimalt otte	ingen grænser for antallet
bredde max. 16 pixels	ubegrænset størrelse
3 farver + baggrundsfarve; men andre farver end de anvendte kan bruges	samme antal som på den anvendte skærm
nøjagtig gengivelse	undertiden flimrer større BOBs

Tabel 4.5: Forskelle på sprites og BOBs

Måske er det påfaldende, at BOBs og sprites stort set er behandlet ens i de hidtidige programmer, men der er intet i vejen for at bruge begge typer bevægelige objekter i samme program.

REM Abschuss
REM P.4.7.-1

I forbindelse med nogle forberedelser fastlægges bølgeformerne for tonekanalerne 1 og 2, så bestemmes de fire farver med PALETTE, og "brugsanvisningen" udlæses. Det er mere interessant i programafsnittet *anfang. En ny sætning indsnævrer objekternes bevægelsesområde:

> **OBJECT.CLIP (x1,y1)-(x2,y2)**
> inskrænker det område, objekterne kan bevæge sig i, til rektanglet med øvre venstre hjørne ved (x1,y1) og nedre højre hjørne ved (x2,y2)

(Normalt er grænserne for bevægelsesområdet lig med output-vinduets mål.) Uden for begrænsningen er ingen BOBs eller sprites synlige. Grænsen gælder desuden som vinduesbegrænsning for COLLISION-sætningen. I programmet *Abschuss spærres det øverste område af vinduet, som teksten viser sig i, for objekterne ved hjælp af denne sætning. Grænselinien standser dog ikke et objekt brat. Afhængig af hvor hurtigt det bevæger sig, trænger det en smule ind over linien og ind i den forbudte zone.

Derefter fastlægges egenskaberne (parametrene) for de tre benyttede objekter. Her møder vi atter en ny sætning:

> **OBJECT.AX objektnummer,accelerationsværdi**
> **OBJECT.AY objektnummer,accelerationsværdi**
> fastlægger accelerationen, den fart hvormed bevægelseshastigheden af et objekt (sprite eller BOB) med betegnelse objektnummer ændrer sig i x- resp. y-retningen.

Amiga-Basic har en kommando, der tillader kontinuerlig acceleration af objekter. Accelerationsværdien indtastes - som hastighed - i form af pixels pr. sekund. I eksemplet får objektet *ang en acceleration på 3 i X-retningen og en på -1 i Y-retningen. Efter at det andet objekt, en angribende raket er startet, kommer vi til en WHILE-løkke.

anfang:
ende:

Først undersøges joystick'ets affyringsknap i WHILE-løkken. Hvis den er nedtrykket, får angrebsraketten sine startværdier og jager hylende opad. Der forgrenes til lydrutinen *rakton. Så vises antallet af træffere og forbiere. Når en af de to værdier har nået ti, forlades sløjfen. Man kan selvfølgelig indtaste et større tal end 10 for at forlænge spillets varighed. Så snart sløjfen forlades, fjernes objekterne fra skærmen og hukommelsen, og programmet slutter med at vise antallet af træffere eller forbiere.

OBJECT.CLOSE objektnummer1,objektnummer2,...
frigør den hukommelse, der optages af et eller flere objekter

Denne kommando må aldrig mangle i slutningen af et program, der bruger BOBs. Den sikrer, at der ikke ødsles med hukommelsen.

Det var et overblik over programfunktionerne, nu kommer vi til den første subrutine *zusammen. Den indholder også en ny funktion, der giver oplysninger om, hvad der skete under en kollision. Amiga-Basic lagrer alle en kollisions-data i en art kø. Hver ny kollision tilføjes til kenø og forbliver der, indtil den hentes med COLLISON-funktionen.

v=COLLISION(n)
alt efter værdien af n overgives der forskellige objekt-parametre

---

COLLISION(0)
nummerbetegnelsen for det objekt, der sidst stødte sammen med et andet objekt. Indholdet af kollisionskøen ændres ikke af dette kald

COLLISION(0)
nummerbetegnelse for det vindue, hvori det med COLLISION(0) fastslåede sammenstød har fundet sted

COLLISION(n0)
Betegnelse for det objekt, der er stødt sammen med objektet n. Samtidig slettes denne information fra køen.

Tabel 4.6: Mulige anvendelser af COLLISION-funktionen

Resultat	Betydning
-1	objektet stødte mod den øvre vinduesbegrænsning
-2	objektet stødte mod venstre vinduesbegrænsning
-3	objektet stødte mod den nederste vinduesbegrænsning
-4	objektet stødte mod højre vinduesbegrænsning
n0	objektet stødte sammen med objekt n

Tabel 4.7: Betydningen af COLLISION-funktionens events.

Normalt henter man informationen om den sidste kollisions-event med COLLISION(0). Man spørger videre med den værdi. Sådan gør det viste program. Der spørges, om et objekt er stødt sammen med vinduesbegrænsningen (c1. Det var et dårligt skud. Andet spørgsmål gælder de to raketter med numrene 2 og 3. Hvis resultatet er positivt, er der sket en affyring af de to raketter. Sådan skulle (!) det i alt fald fungere.

I praksis gælder desværre, at kollisions-undersøgelsen går for langsomt, sammenlignet med den fart, sprite'ne kan fare over skærmen. I programmet besværliggøres det af, at træfferne registreres i et vindueshjørne. Der sker altså hele tiden tre kollisions-events på en gang. Vil man have bedre træffer-resultater, bør man mindske accelerationen af objektet *abw. Nu forstår man, hvorfor der to gange blev hentet værdier fra kollisionskøen ved *anfang. Den ene var til at fjerne en (uinteressant) event fra køen.

zusammen:
fehler:
konstruieren
speichern:
spr:
rakton:
rakton1:
puls:

Sidst kommer rutinerne til konstruktion og gemning af objekter.

Se på kontrasten mellem BOBs og sprites i begyndelsen af kapitlet med "*Abschuss". Her har vi to BOBs, angrebsraketten og den bevægelige affyringsrampe, og én sprite, forsvarsraketten. En sammenligning af hastigheden falder ud til fordel for sprites; men BOBs vinder med hensyn til længde. Det passer ikke, at sprite-størrelsen er begrænset til 16 pixels. Kig en gang på forsvarsraketten. Den er dobbelt så bred, 32 pixels. For at demonstrere det endnu tydeligere, har jeg indbygget en linie i programmet, der ikke har noget

at gøre med programkørslen. En linie før den indre WHILE-løkke trækkes en linie, der ligger lige under spidsen af forsvarsraketten og er 16 pixels bred. Hermed er det klart, at denne sprite er bredere end 16 pixels. Manualen har alligevel ret i, at en sprite kun er 16 pixels bred; disse pixels vises blot i dobbelt bredde, når man anvender en skærm med 640 pixels pr. linie.

Endnu et punkt ved sammenligningen var farvegengivelsen i de to typer objekter. Lad programmet køre og se på forsvarsraketten. Selv om der kun anvendes en skærm med fire farver, udviser sprite'n tre ekstra farver, det kan BOBs ikke.
Bevægelsen af objekter på skærmen kaldes "animation"; men egentlig betyder ordet "levendegørelse". Det eneste vi så, var en mere eller mindre elegant forskydning af uforanderlige genstande. Ægte animation forklares i følgende afsnit.

Start Objekt Editoren og vælg sprite-editering. Indstil sprite-højden til 31 pixel og omstil til 4x4-forstørrelse. Forsøg at tegne en lille mand set fra siden. Det er bedst at tegne de to ben i forskellig farve. Tænk på, at sprite'n bliver dobbelt så bred på skærmen - tegn manden så slank som muligt. Når man er tilfreds, gemmesmanden som "fg1". Nu loader man igen billedet med kommandoen Open. Lav om på figuren ved at ændre på benstillingen. Ialt laver vi fem forskellige modifikationer, der viser manden i fem forskellige stadier af gang, og gemmer dem som "fg2" til "fg6".

Gå ud af Objekt-Editoren og læg de seks ikoner og spritene sammen med det følgende program i en anden skuffe. (Hvis man ikke kan se ikonerne med det samme, lukker og åbner man bare vinduet BASICDemos.)

REM Bewegung
REM P.4.7-2

Start programmet og kig på rigtig animation. Den er måske ikke perfekt, men kan forbedres ved at ændre sprite'ne. Programmet er ret enkelt. Fremmed- og egenværdierne af alle objekter sættes til nul med OBJECT.HIT. I teorien skulle kommandoen være overflødig, da der ikke bliver undersøgt for kolisioner. I praksis forhindrer det en let rykken i sprite'ne, der undertiden kommer til at overlappe. I WHILE-løkken sættes sprite'ne i bevægelse i rækkefølgen fg1 til fg6 og ind- og udkobles cyklisk. Det hurtige skift mellem sprites kommer til at se ud, som om spriten ændrer eller bevæger sig. Der er en ny funktion, der sørger for, at bevægelsen overgives gnidningsfrit fra den ene sprite til den anden.

v = OBJECT.VX (objektnummer)
v = OBJECT.VY (objektnummer)
giver hastigheden af objektet (sprite eller BOB) med betegnelsen objektnummer i pixels pr. sekund.

I programmet benyttes funktionen, så hastigheden, af den foregående sprite i rækken, bliver anvendt som hastighed for den nye sprite. Det kan f.eks se sådan ud:

OBJECT.VX x2, OBJECT.VX(x3): REM x2 får x3's hastighed

Der findes endnu et par lækkerbiskener i Kapitel 5.6

4.8 Så ruller skærmen

Når man viser programmer i List-vinduet, forekommer det naturligt, at skærmindholdet kan bevæges i alle fire retninger - det kan man gøre med Workbench-skærmen. Det er dog ikke altid. På mange computere kan billedet kun bevæges opad eller nedad (man kalder det også "rulle" eller "scrolle"). Amigaen har en SCROLL-kommando, man kan benytte i egne programmer.

SCROLL(x1,y1)-(x2,y2),X,Y
forskyder indholdet af rektanglet (x1,y1)-(x2,y2) i det aktive vindue X punkter mod højre og Y punkter nedad.

REM Roll-Test
'P.4.8-1

Vi har lært så meget, at scroll'ningen går så hurtigt, at man er nødt til at bremse det i et praktisk program, fordi man ellers ikke får noget at se. Før vi prøver det, bør læseren eksperimentere lidt med det første program. Lav om på værdierne af forskydningen i X- og Y-retningerne. Man vil se, at i nogle retninger bliver forskydningen inden for det angivne område, mens der i andre forskydes uforudsete billedudsnit med.

Desværre er SCROLL begrænset til det synlige billedområde. Man kan ikke forskyde et større område, end der er fastsat af vinduesstørrelsen.

Nu til den forsinkelse af scroll-bevægelsen, som er uundværlig til mange formål. Man opnår den lettest ved kun at foretage små bevægelser i den ønskede retning og eventuelt indlægge pauser.

REM Hochhaus
'P.4.8-2

Forskydningen af højhuset nedad er i dette eksempel opnået med en FOR-løkke. Bevægelsen i Y-retningen sker via sløjfetællerens variabel i. Herigennem opnås en acceleration af SCROLL-processen, fordi variabelens værdi stiger kontinuerligt.

Man kan ikke hente huset tilbage med en modsat scroll-bevægelse. Et område, der er forskudt, kan ikke hentes tilbage, når det er blevet usynligt. Har man ikke scrollet længere, end at det tegnede billede kan ses, kan det godt lade sig gøre.

REM softball
'P.4.8-3

Dette simple program antyder de muligheder, der er i SCROLL-kommandoen. Prøv at ændre sløjfetælleren til

```
FOR j = 14 TO 1 STEP -2
```

og

```
FOR i = 1 TO j
```

eller at ændre SCROLL-sætningens j til j/2 og angivelserne i CIRCLE- og PAINT-sætningerne til (20,30). Bolden hopper op ad bjerget, og dens hop bliver mindre og mindre, indtil den til sidst ligger helt stille. Det næste programeksempel, der også udnytter SCROLL-sætningen intensivt, flytter os til Cape Canaveral (eller Baikonur).

REM Fehlstart
'P.4.8-4
Raketten ser lidt komisk ud, men er en smuk demonstration af PATTERN-komandoen, som vi hidtil kort har strejfet.

PATTERN liniemønster, feltmønster
fastlægger et eller to mønstre, der bliver brugt af Amiga-Basic til tegning af linier, resp. udfyldning af flader i stedet for en "solid" flade

Herunder skal liniemønster angives i form af et kort heltal. De 16 bits fastlægger, hvilke punkter der bliver tegnet på linien, og hvilke der forbliver uberørt. Hver sat bit (lig med 1) i liniemønster betyder, at et punkt i linien tegnes.

Som feltmønster skal angives et heltalsfelt, der indeholder udfyldningsmønstret. Værdien af hvert element i feltet opfattes som en række på 16 bits, der hver beskriver en linie af mønsteret. En sat bit i elementet svarer til et farvet punkt i mønstret. Bits (punkter), der ikke er sat, forbliver uændrede. Et mønsters bredde er: 16 pixels. Højden afhæger af feltets længde. Den kan være alle potenser af to (1, 2, 4, 8, 16, 32 osv.).

Mønstre skal altså altid angives i form af tal. Decimaltal er dårligt egnede, fordi det er svært at se forholdet mellem dem og de tilsvarende bits. Anderledes med hexadecimal- og oktaltal, som Amiga-Basic også kan arbejde med.

Betegnelse	Cifre	Antal	Kendetegn	Eksempel
oktaltal	0 til 7	max. 6	&O eller &	&O777777
hexadecimaltal	0 til F	max. 4	&H	&HFFFF

Tabel 4.8: Oktal- og hexadecimaltal

Hexadecimaltal svarer til følgende decimaltal:

Hexadecimalt	Decimalt
0 til 9	0 til 9
A	10
B	11
C	12
D	13
E	14
F	15
10	16
11	17
...	...
FFFF	65535

Tabel 4.9 Hexadecimaltal-cifrene og deres betydning

Amiga-Basic er ligeglad med, hvilket talsystem, man angiver tallene i. Man skal med kendetegnet vise, hvilket system, man anvender. Man kan bruge flere forskellige talsystemer i samme formel.

decimalt	?SQR(30625)	resultat: 175
oktalt	?SQR(&73641)	resultat: 175
hexadecimalt	?SQR(&H77A1	resultat: 175

(SQR beregner kvadratrod.) Alle tre rækker cifre i eksemplerne svarer til samme tal og giver samme resultat. Kun beskrivelsen af tallene er forskellig. Resultatet af en beregning gives dog altid i decimaltal.

Prøv to eksempler til

decimalt	?345/58*89	resultat: 529.3965
blandet	?345/&H3A*&131	resultat: 529.3965

Som det ses, kan man principielt til enhver tid anvende det talsystem, der forekommer mest egnet. Tallene kan ikke blot indlæses i forskellige talsystemer, men også udlæses på samme måde. Hertil tjener to funktioner, der forvandler en værdi til en streng, som indeholder tallet i det ønskede talsystem.

v$ = OCT$(numerisk udtryk)
giver en streng, der repræsenterer værdien af numerisk udtryk i form af et oktaltal

v$ = HEX$(numerisk udtryk)
giver en streng, der repræsenterer værdien af numerisk udtryk i form af et hexadecimaltal

Ingen af funktionerne tager hensyn til eventuelt tilstedeværende decimaler, men omdanner kun heltalsdelen af et tal. For fuldstændighedens skyld resumerer tabellen alle omregningsmuligheder.

Omregning	Basic-kommando
decimal til hexadecimal	PRINT HEX$(decimaltal)
hexdecimal til decimal	PRINT &H hexadecimaltal
decimal til oktal	PRINT OCT$(decimaltal)
oktal til decimal	PRINT &O oktaltal
oktal til hexadecimal	PRINT HEX$(&O oktaltal)
hexadecimal til oktal	PRINT OCT$(&H hexadecimaltal)

Tabel 4.10: Omregninger fra det ene talsystem til det andet

REM Muster
'P.4.8-5

Programmet viser to linier med forskellige liniemønstre og udfylder tre geometriske objekter med to forskellige fladmønstre. Vi ser på liniemønstret. Det fastlægges af PATTERN &H9249. Det tilsvarende 16-bit tal ser sådan ud:

hexadecimaltal	9	2	4	9
bitrække	1001	0010	0100	1001
decimalværdi	2^15+2^12	2^9	2^6	2^3+2^0
				= 37 449

På de steder, hvor der står et 1 i masken, er bit'en sat. Dér afsættes også et punkt. Den linie, der frembringes af PATTERN-sætningen, ser derfor således ud:

punkt,tom,tom,punkt,tom tom,punkt osv.

For at beregne den tilsvarende decimalværdi, skal man sammentælle de enkelte værdier, som bit'ene svarer til:

2^15+2^12+2^9+2^6+2^3+2^0.

Det giver 37449, som ikke har ret megen lighed med liniens synlige stregmønster, og kræver en del regneri. Det er lettere med hexadecimaltallene. Her opdeles mønstrets 16 bits i 4 "nibbles", grupper på fire bits. I hver nibble adderes separat de til de fire bits svarende decimaltal (8, 4, 2 og 1), og det resulterende højst tocifrede decimaltal (mellem nul og 15) forvandles til ét hexadecimalt ciffer (mellem 0 og F).

	4. nibble	3. nibble	2. nibble	1. nibble
Bitmaske	1 0 0 1	0 0 1 0	0 1 0 0	1 0 0 1
Decimalsum	9	2	4	9
Hexadecimalt ciffer	9	2	4	9

Af en bitrække kan man uden komplicerede beregninger danne det tilsvarende hexadecimaltal - eller det er i alt fald meget lettere end at finde ud af decimaltallet. For at tydeliggøre processen, skal den vises med det andet liniemønster. Det giver en ubrudt linie (uden uberørte punkter).

	4.Nibble	3.Nibble	2.Nibble	1.Nibble
Bit-Maske	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
Summen dezimal	16 16	16	16	
Hexadezimalziffer	F F	F	F	

Nu, hvor vi ved, hvor let det er med liniemønstre, kan vi beregne felter til udfyldte flader. Her kan man tage et stykke ternet papir til hjælp. Tegn først det pilmønster, der i programmet bruges til at fylde rektanglen. Det er defineret i den første PATTERN-sætning. For at gøre det overskueligt er ikke satte bits tegnet som et punktum og satte bits som et x i den følgende tabel.

	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	
0. Feld-Element	. . . x	. . . .	. . . x	. . . .	&H1010
1. Feld-Element	. . x x	x . . .	. . x x	x . . .	&H3838
2. Feld-Elemen	. x x x	x x . .	. x x x	x x . .	&H7C7C
3. Feld-Element	x x x x	x x x .	x x x x	x x x .	&HFEFE

Og allerede har vi konstrueret vor første udfyldte flade. Fordi det var så let, går vi straks i gagn med bilen. Mønsteret til den er otte elementer højt. Vi erindrer, at man kun kan benytte tal, der er potenser af to, til mønstrets højde. For at vise, at konstruktonen er endnu lettere med oktaltal, er begge muligheder angivet i den følgende konstruktion.

hexadezimal	8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1	
0. Feld-Element	. . . .	&H0 &O0
1. Feld-Element	. . . . . . x x x x x x x . . .	&H3F8 &O1770
2. Feld-Element	. . . . . x x . . x . . x x . .	&H64C &O3114
3. Feld-Element	. . x x x x x . . x . . x x x x	H3E4F &O37117
4. Feld-Element	. x x x x x x x x x x x x x x x	&H7FFF &O77777
5. Feld-Element	. x x x x x x x x x x x x x x x	&H7FFF &O77777
6. Feld-Element	. . . x x . . . . x x . .	&H180C &O14014
7. Feld-Element	. . . .	&H0 &O0
oktal	1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1	

Fatsættelsen af hvilke oktaltal, der hører til en bitrække, fungerer ligesom med hexadecimale tal. Blot opdeles 16-bittallet i fem grupper på tre bits (og én enkelt bit).

Hvilket talsystem man end vælger, så kan man opnå mange effektfulde billeder med PATTERN-sætningen. Lad os nu vende tilbage til SCROLL-kommandoen. Efter de simple eksempler i starten af kapitlet er her et mere anvendeligt. Det kan udmærket bruges til indledning af egne programmer.

REM Vorspann
'P.4.8-6

Programmet gir et imponerende indtryk, og dog benytter det, bortset fra en enkelt superrutine, ikke andet end SCROLL-kommandoen. Alligevel en kort forklaring. Først fyldes skærmen helt af et rødt rektangel, og så tegnes der nogle stjerner i dette rektangel. Stjernen konstrueres i underprogrammet *stern. Så indlæses DATA-sætningerne, og rektanglet rulles op som tæppet i et teater. Dernæst følger udlæsningen af en tekst, der "ædes op" af en samtidig scrollning nedefra og opad. Det samme sker med næste tekst, blot i vandret retning. Så skylder jeg blot at forklare rutinen *zeichnen. Den kan findes i afsnit 5.2.

KAPITEL 5 TIPS OG TRICKS

Kapitel 5 er en - ret rodet - samling af emner.

Nogle af Amigaens Basic-kommandoer skriger på at blive brugt til noget, som de ikke er beregnet til. Så bliver det rigtigt interessant for en programmør. Den art manipulation med kommandoer behandles her.

Og endelig er der nogle kommandoer, der ikke passer ind i et bestemt kapitel og derfor gennemgås her.

5.1 Disk uden diskette

RAM-disk! Det er en art diskette, der ikke er nogen rigtig diskette, den simuleres af software, idet et bestemt RAM-afsnit reserveres og behandles som en diskette. Fordi der ikke er nogen bevægelige mekaniske dele, er en sådan "diskette" ulige hurtigere end en rigtig diskette. Men den lider også af en stor mangel: data, der er opbevaret på den, eksisterer ikke længere, efter at computeren er slukket eller systemet er crashet! Hvis man vil udnytte RAM-diskens hurtige arbejdsmåde og alligevel vil bevare dataene på den, skal man huske at kopiere dem over på en rigtig diskette, før man slukker.

For at se, hvor hurtig en RAM-disk er, skal man bruge det følgende program.

REM RAMDisk
'P.5.1-1

I programmet gemmes samme tekst én gang på den normale diskette og én gang på RAM-disken. Før der begyndes at skrive, fastholdes den aktuelle tid i variablen v1 ved hjælp af TIMER-funktionen, og når processen er afsluttet, gemmes den aktuelle tid igen i variablen v2. Forskellen på de to værdier er altså den tid, som skriveprocessen tog. Programmet gør det samme under læseprocessen. Følgende tabel viser de værdier, som jeg fandt herunder (forskellige Amigaer kan vise lidt afvigende værdier).

	Gemme	Læse
Normal diskette	7.50 sec	1.05 sec
RAM-disk	0.18 sec	0.09 sec

Tabel 5.1: Tidsforskelle på diskette og RAM-disk

Som der ses, er det ganske pæne forskelle. Det næste program gemmer grafik på RAM-disken og indlæser den atter. Læg mærke til den fart, det hele sker med.

REM Adventure
'P.5.1-2

Navnet "Adventure" er lidt af en overdrivelse til dette program. Men indlæsningen af billeder til et rigtigt adventure program kunne udmærket ske på den måde. I den første del af programmet tegnes nogle enkle billeder til demonstrationsformål. Her kan man også indsætte billeder, som er tegnet med PAINTAmiga (eller Deluxe paint eller Graficraft osv.). Så gemmes billederne på RAM-disken. Det egentlige program begynder ved hopmærket *spiel. Her kan man komme ind i de billeder, der angives af tallene. Det varer kun omkring ti sekunder at indlæse et 50 linier højt billede fra RAM-disken. Tænk på den tid, Basic, resp. rutinen *iffLesenSpeichern, bruger til at afkode billedet.

5.2 Stor, større, størst

Følgende program viser, hvordan man får ekstremt store bogstaver på skærmen. Når man har prøvet det, finder man sikkert let på anvendelsesmuligheder. Det er f.eks udmærket til at lave introer til programmer. Som de øvrige programmer er det beregnet på en liniebredde på 80 tegn.

Efter starten spørger programmet om den forstørrelse, der ønskes. Der kan indtastes tal mellem 1 og 20. 1 giver ingen forstørrelse, 2 giver en fordobling af tegnets højde og bredde, 3 en tredobling osv. lige indtil 20, der giver gigantiske bogstaver.

REM Das grosse ABC
'P.5.2-1

Efter at have valgt forstørrelse, kan teksten indtastes normalt via tastaturet. Da hvert tegn lagres i tastaturbufferen, behøver man ikke at afvente forstørrelsen, men kan straks skrive næste bogstav. Når skærmen er fuld, eller man trykker RETURN, afsluttes programmet.

Programmets arbejdsmåde er i princippet meget enkel. Først tegnes bogstavet i normal størrelse på skærmen i et bestemt rektangel, som aftastes punkt for punkt med POINT. Hvert sat punkt giver så et lille (eller stort) rektangel i den forstørrede gengivelse.

Vil man bruge programmet som indledning til noget større, skal tastaturindlæsningdelen fjernes. Så kan man bedre lægge den tekst, der skal vises, bagest i DATA-sætninger og indlæse den fra dem.For at tilsløre at de store bogstaver frembringes ud fra små, kan man sætte bogstaverne usynligt. Det kræver, at man forbigående sætter en farve lig baggrundsfarven med PALETTE-sætningen og viser bogstaverne med den:

PALETTE 0, 0, 0, .6 : PALETTE 2, 0, 0, .6 : COLOR 2

5.3 Specialtegn

I Basic-manualens tabel over specialtegn mangler nogle interessante koder, som jeg ikke vil snyde læseren for.

5.3.1 Tegnkoder ved tastaturaflæsning

I følgende oversigt er de enkelte tegnkoder for overskuelighedens skyld kun anført med deres cifre. CHR$(n) giver deres betydning. I programmer kan de aflæses på følgende måde:

```
tast:
ta$ = INKEY$
IF ta$ = "" THEN tast
IF ta$ = CHR$(n) THEN forgrening
GOTO tast
```

Her er n et ciffer fra en af de følgende tabeller, og forgrening angiver en subrutine. For at lette forståelsen er der et eksempel i slutningen af dette kapitel.

Kode	Betydning	Kode	Betydning
8	BACKSPACE		
9	TAB		
13	RETURN		
27	ESC		
28	cursor øverst		
29	cursor nederst		
30	cursor til højre		
31	cursor til venstre		
32	mellemrumstangent		
127	DEL	31	CTRL+DEL
129	F1	1	CTRL+F1
130	F2	2	CTRL+F2
131	F3		
132	F4	4	CTRL+F4
133	F5	5	CTRL+F5
134	F6	6	CTRL+F6
135	F7	7	CTRL+F7
136	F8	8	CTRL+F8
137	F9	9	CTRL+F9
138	F10	10	CTRL+F10
139	HELP	11	CTRL+HELP

Tabel 5.2: ASCII-koder

5.3.2 Tegnkoder til PRINT-sætningen

Ligesom man meddeler et program tryk på bestemte taster i form af ASCII-koder, kan det også udlæse tegn, der ikke kommer på skærmen som bogstaver, men giver specielle effekter. Sætningen PRINT CHR$(7) giver en tone og blinker med skærmen. Denne PRINT-sætning svarer altså til et kald af BEEP-kommandoen.

Kode	Virkning
7	som BEEP-sætningen
8	som BACKSPACE-tasten
9	som TAB-tasten
10	som PRINT-sætningen alene (liniefremføring)
12	som CLS-sætningen

Tabel 5.3: Specielle ASCII-koder til udlæsning med PRINT

Endnu et problem, som vi ofte møder, er de danske "specialtegn".Den følgende tabel lister deres ASCII-koder:

Kode	Bogstav	Kode	Bogstav
190	æ	174	Æ
191	ø	175	Ø
140	å	129	Å

Tabel 5.4: ASCII-koder for de danske specialtegn (bogstaver).

Det følgende program demonstrerer, hvordan de danske specialtegn kan benyttes i et Basic-program. (Amiga-Basic reagerer meget uvenligt på disse tegn - undgå altid at bruge dem i variabelnavne eller lignende i et Basic-program.)

REM Staedte raten
REM P.5.3-1

Et gætteprogram af den type kan udformes mere elegant, men det er fint til at vise anvendelsen af specialtegnene. Hvis man alligevel vil bruge det som spil, skal man blot udvide DATA-sætningerne. Opbygningen af disse linier er således: første ord i en DATA-sætning udlæser landet, og de følgende fire ord er steder i det pågældende land. Tallet sidst i DATA-linien er ASCII-koden for den funktionstast, man skal trykke på. Desuden skal dimensioneringen af feltet J$ og angivelserne i de to FOR-løkker forhøjes tilsvarende, når man udvider spillet.

5.4 Tilfældet

I nogle anvendelser - særligt ved programmering af spil - er tilfældighed nødvendig. Hertil bruges funktionen RND (engelsk random = tilfældig).

v=RND
giver et tilfældigt tal mellem 0 og 1

Prøv funktionen for at få et indtryk af tilfældigheden i den talrække, der fremkommer ved gentagne kald.

```
FOR i = 1 TO 10: PRINT RND: NEXT
```

Hvis man bliver ved med at indtaste denne linie, vil man se, at man hver gang får samme talrække. Tit er det ikke særligt generende. Hvis det er vigtigt at få forskellige tilfældige tal ved hvert gennemløb, skal man anvende en funktion til.

RANDOMIZE n
RANDOMIZE TIMER
starter tilfældighedsgeneratoren med en ny begyndelsesværdi

Herunder er det nødvendigt at opgive et tal eller TIMER. Hvis det mangler, stopper Basic og beder direkte om et tal.

Random number seed (-32768 to 32767)?

Den bedste metode til frembringelse af tilfældige tal er ellers RANDOMIZE TIMER. Hvis man angiver et tal i stedet for TIMER, giver tallet altid samme række tilfældige tal. RANDOMIZE 2 giver altså en anden række tal end RANDOMIZE 3.

Det er ofte nødvendigt med tilfældige tal i et bestemt område, men RND giver kun tilfældige værdier mellem 0 og 1. Det kan ændres med en simpel formel. Hvis vi ønsker tilfældige tal mellem to værdier min og max, giver følgende formel de ønskede tal:

RANDOMIZE TIMER: z = min + RND * (max-min)

Vil man kun have heltal, kan man få det med INT-funktionen. Prøv med et eksempel. Det følgende program giver tilfældige tal mellem 1 og 49, som i Lotto.

REM Lotto
'P.5.4-1

I det andet eksempel med tilfældige tal, kan man spille terninger mod Amigaen. Da de tilfældige tal også i det tilfælde er opnået via RANDOMIZE TIMER, er chancerne absolut ens; Amigaen spiller helt fair.

REM wuerfel
REM P.5.4-2

5.5 At stå på hovedet

I begyndelsen af Kapitel 2 lærte vi, at GET-sætningen er en af de mest alsidige grafik-sætninger; men har du opdaget hvordan sætningen opbevarer de data, den har læst fra skærmen? Man kan gøre noget med det numeriske felt, der benyttes til lagring af billeddataene - hvis man kender formatet nøjagtigt.

REM Get-manipulation
'P.5.5-1

Der tegners et lille kvadrat, der med GET hentes til det numeriske felt *bild. Hvis rektanglet atter sættes på skærmen med PUT, er der pludselig blevet en stige ud af det. Årsagen ligger i FOR-løkken, sætter nogle elementer til nul i trin på to. Det antyder de muligheder, som en sådan felt-manipulation giver.

I eksemplet blev billedinformationen lagt i et felt med korte heltal. Et kort heltal er 2 bytes (16 bits) langt. Hver bit heri svarer til et enkelt billedpunkt (pixel). Det betyder, at man kan lagre billedudsnit på 16 pixels bredde og en pixels højde i et feltelement. Lad os se, hvordan de første feltelementer ser ud. (Man kan finde ud af det ved at lade felterne udlæse med PRINT.)

0. element	16 bits	0000000000001111	=15	bredde
1. element	16 bits	0000000000111111	=63	højde
2. element	16 bits	0000000000000010	= 2	bitmap-dybde
3. element	16 bits	1111111111111110	=-2	15 pixels sat
4. element	16 bits	0000000000000000	= 0	alle pixels slettet
5. element	16 bits	1111111111111110	= 2	15 pixels sat
6. element	16 bits	0000000000000000	= 0	alle pixels slettet
7. element		...		...

De første tre feltelementer indeholder information om bredde, højde og dybde. De næste elementer indeholder så de egentlige billeddata, altså bits, der angiver, om et punkt er sat eller ej. I ovenstående eksempel er elementerne sat lig nul og de tilsvarende pixels forsvundet.

Dette enkle format passer kun, når programmet kun skal arbejde med to farver, hvad der sjældent er tilfældet. Allerede Worbench-standardskærmen benytter fire farver. Det gør det hele lidt mere kompliceret. Det tredje feltelement

indeholder værdien af dybden 2. Det betyder, at for hvert skærmbilledpunkt ligger der to bits oven på hinanden, som svarer til fire farver.

Første bit	Anden bit	Farve		
0	0	blå	=	farve 0
1	0	hvid	=	farve 1
0	1	sort	=	farve 2
1	1	rød	=	farve 3

Tabel 5.5: Sammenhæng mellem bits og farveværdier

De nævnte farver er standardfarverne. Kun nummeret har betydning for tildelingen af bits til en bestemt farve. Da der kræves to bits til hver pixel, rejser det spørgsmålet om, hvor i det af GET frembragte felt de to bits står - de står nemlig ikke efter hinanden. Heldigvis er formatet simpelt. Først kommer alle data i det første bitplan, så alle i det andet bitplan (og i det tredje, hvis der benyttes otte farver, og så fremdeles). Man kan let kontrollere, om informationen er rigtig. Lav farven i LINE-sætningen fra programmet "GET-manipulation" om fra 1 til 3. Så fremkommer stigen i rødt og sort, fordi rødt betyder, at alle bits i begge bitmaps er sat (farvetal 3), medens sort betyder, at der kun er sat én bit i ét bitplan. Pøv det! Hvis man ikke er overbevist om, at den sorte farve ses dårligt på mørkeblå baggrund, kan man tilføje følgende linie bag linien med DIM...

PALETTE 0,1,1,0

Dermed har vi ændret baggrundsfarven fra blå til gul og den sort-røde stige er let at se.

Hvad kan man gøre med denne nyerhvervede viden? Man kan f.eks. stille et billede på hovedet! Det kræver, at man finder ud af, hvordan de enkelte felter skal omfordeles. Det er let nok: man skal ombytte sidste linie i feltet med den første, næstsidste med den anden osv. Ombytningen kan udføres separat for hvert af de to (eller flere) bitmaps. Problemet består i at beregne, i hvilket felt linien af det pågældende bitplan begynder. Det følgende program klarer opgaven. Det kopierer den øverste halvdel af skærmen til den underste og spejlvender samtidig billedet.

```
REM Fatamorgana
REM 'P.5.5-2
```

I første del forsynes variablerne med værdier. De har udførlige navne, der er forklaring nok på, hvad der indeholdes i variablerne, resp. hvorfra værdierne stammer. Billedet opstår i rutinen *zeichnung. Man kan også indsætte egne tegninger eller tekster her. Rutinen *kippen ombytter i to parentetiske FOR-løkker de to bitplaner i et billede, der er læst med GET, som lige beskrevet. Vil man kippe hele billedet, skal tre linier ændres.

```
hoejde = hoejde/2          slet
CLEAR, 35000               til CLEAR,60000
PUT (0.90),nytbillede      til PUT (0,0), nytbillede
```

5.6 BOBs og sprites på en anden måde.

I kapitlerne om BOBs og sprites er vist, hvad man kan gøre med bevægelige objekter. Der er flere tricks og kneb, som vi skal se nogle af i dette kapitel. "ObjEdit" anvender faste farver, nemlig hvidt, sort og rødt, objekt-editoren kan ændres, så man kan anvende andre farver. Ønskes man at ændre sprite-farve, kan man lade et program klare sagen.

Det kræver, at man kender opbygningen af den streng med sprite-data, som sådan en fil indeholder. Følgende program indlæser disse data og udlæser dem så, som en række tal på printeren. (Hvis man i stedet vil se dem på skærmen, skal man udskifte LPRINT med PRINT.) Senere kan man anvende de udprintede tal til at gemme sprite'n eller BOB'en som DATA-sætninger og behøver ikke at indlæse den fra disketten ved hver start.

REM Objekt-Daten
'P.5.6-1

Bag start spørgers, om dataene skal udprintes. Så overgives navnet på det ønskede objekt til programmet med endnu en INPUT-kommando. Her kan man angive filer, der indeholder både sprites og BOBs. Efter åbningen af filen, fastslås dens læmgde først med LOF.

v=LOF(filnummer)
giver længden af filen med betegnelsen filnummer i bytes

Ud fra længden kan man fastslå objektets antal dataværdier. Den hertil anvendte formel forstås let, når man har læst de følgende forklaringer. Med den fundne værdi n har programmet samtidig det rigtige antal elementer til dimensionering af feltet *sprite

Vil man vide, hvordan et objets tegnkæde er sammensat, ses efter der, hvor tegnkæden frembringes, i programmet "ObjEdit".

1. værdi: ColorSet=CVL(INPUT$(4,1)) er altid nul
2. værdi: DataSet=CVL(INPUT$(4,1)) er altid nul
3. værdi: dybde=CVL(INPUT$(4,1)) er altid to (farvedybde)
4. værdi: bredde=CVL(INPUT$(4,1)) er altid 16 (spritebredde)
5. værdi: højde=CVL(INPUT$(4,1)) er variabel (spritehøjde)

Så følger tre værdier, der hver består af 2 bytes, altså er korte heltal:

6. værdi: Flags=CVI(INPUT$(2,1)) er altid 25
7. værdi: PlanePick=CVI(INPUT$(2,1)) er altid tre
8. værdi: PlaneOnOff=CVI(INPUT$(2,1)) er altid nul

(Når der står "er altid..." gælder det kun for de her omtalte tilfælde.) Fra den 9. værdi følger igen værdier på to bytes, til filen er slut. Den 9. værdi er den første virkeligt vigtige. Her begynder de data, der indlæses med en GET-sætning under sprite-programmeringen. Dataene i området kan manipuleres, som det blev vist i kapitlet "på hovedet"; men for dette program er de sidste tre værdier endnu mere interessante. De svarer ikke mere til GET-sætningens feltindhold og indeholder sprite'nes tre farver.

Hver af de tre farver gemmes som et 16-bit tal, hvoraf kun de højre 12 bits udnyttes. Hver farve er sammensat af den røde, den grønne og den blå komponent, og for hver af disse er der en nibble (4 bits) af 16-bit tallet til rådighed (= 12 bits i alt).

	Røde nibble	Grønne nibble	Blå nibble
værdi:	2048 1024 512 256	128 64 32 16	8 4 2 1
bitnummer:	11 10 9 8	7 6 5 4	3 2 1 0

Tabel 5.6: De tre komponenter af en farve, fordelt på tre nibbles

Hver nibble kan have 16 forskellige værdier (mellem 0 og 15, resp. 0 og F hexedecimalt), der svarer til de 16 mulige intensiteter af en farvekomponent. Divideres værdien med 16, får man de intensitetsværdier, der er anvendt i PALETTE-sætningen. Hermed har vi grundlaget for at kunne skrive følgende program, der kan ændre farveangivelserne i en spritefil.

```
REM Sprite-Farben
'P.5.6-2
```

Der spørges, om sprite-værdierne skal printes. Derefter indlæses sprite'ns gamle og nye navn med INPUT. (Den ændrede sprite skrives tilbage på disketten, under det nye navn.) Hvis det gamle navn også opgives som nyt navn, lagres de nye værdier under det gamle navn, og den gamle version er dermed tabt. Så indlæses den sprite, der skal ændres, i feltet sprite. Nu skal der indtastes tre sprite-farver som en række af tre RGB-værdier for hver, nøjagtigt som ved PALETTE-sætningen. De enkelte komponenter omregnes som beskrevet, og overgives til feltvariablen sprite.

Ønskes et udprint af tallene til anvendelse i DATA-sætninger, sker det nu. Herefter lagres de ændrede sprite-data på diskette; og for at kunne se resultatet af ændringen, sættes både den gamle og den nye sprite på skærmen og betegnes tilsvarende. Det er bedst at prøve programmet på sin yndlingssprite.

Ændringen af objektfarverne er ikke kun mulig med sprites. BOB-farverne tages jo fra den aktuelle palet, og kan ændres med den. Amiga-Basic har en særlig sætning til at ændre en BOBs farve senere (men kun inden for de rammer, den aktuelle palet giver).

> **OBJECT.PLANES objekt,bitplan,plan-ind-ud**
> fastlægger, på hvilke bitplaner af billedet BOB'en med betegnelsen objekt "virker", når den ses på skærmen.

En BOB er et lille billede, der forbigående erstatter tilsvarende dele af skærmbilledet. Men hvis BOB'en består af færre bitplaner end skærmbilledet, kan man vælge på hvilke, BOB'en skal afbildes. En firfarvet BOB på en 32-farvet skærm kan benytte fire tilfældige farver ud af de 32, og ikke kun de første fire. (Hvis skærmen og BOB'en har samme antal bitplaner, er der ikke megen mening med OBJECT.PLANES.)

Hvorledes de to tal bitplan og plan-ind-ud påvirker hinanden, er svært at forklare uden yderligere informationer. Ved anvendelse i praksis er det bedst at teste farveændringen på et eksempel. Konstruer en farve-BOB i 8-farve mode til dette formål. BOB'en skal bestå af otte kvadrater over hinanden i farvenummerenes rækkefølge. Kald denne BOB "famu". Det følgende program vil vise, hvordan farvelægningen ændres af OBJECT.PLANES.

REM BOB-farver
REM P.5.6-3

BOB'en tegnes med uændrede farver og ved siden af den samme BOB med farverne modificeret af OBJECT.PLANES. Når der trykkes en tast fremkom-

mer en ny kombination af bitplan og plan-ind-ud, hvis værdier hele tiden vises på skærmen.

Læseren har sikkert i professionelle spil oplevet, hvordan sprites eller BOBs opløses (eventuelt som ved en eksplosion). Der er flere muligheder til at realisere en sådan opførsel i Amiga-Basic. En af dem består i at programmere fem eller ti BOBs, resp. sprites, med de enkelte stadier af ødelæggelsen og vise dem efter hinanden. Det er ret besværligt og kræver nogen programmering. En anden mulighed er at slette de pågældende objektdata lidt efter lidt. Det kræver ingen ekstra objekter, blot et lille stykke program.

REM aufloesen
'P.5.6-4

Ved start spørges efter navnet på en BOB eller sprite, og objektdataene læses fra disketten. For at kunne se, hvordan det intakte objekt ser ud, vises de af *altsprite. Så frembringes der ti nye strenge til de enkelte opløsningstrin. Derfor lagres de første fem som lange heltal fra oprindelsesobjektet i strengen *te1$. Så bliver objektdataene sat til nul i skridt på ti, idet der begyndes fra den elvte værdi. Herunder gøres en undtagelse med de sidste værdier, for at farverne skal bliver slettet ved en fejltagelse (ved sprites). De frembragte sprite-data tildeles til et element af feltvariablen te$. Sletningsprocessen gentages ialt ti gange, så feltet derefter indeholder 10 forskellige objekter i forskellig grad af opløsning.

Den hidtil omtalte programdel egner sig godt til anvendelse i selvkonstruerede programmer. Så har man ti nye objekter til en tilsvarende begivenhed. Resten af rutinerne demonstrerer den tilsigtede effekt og skal ikke medtages i andre programmer. I *neusprite fjernes det gamle objekt fra hukommelsen med OBJECT.CLOSE. Istedet fremkommer de ti delsprites efter hinanden (på samme position); således opløses det oprindelige objekt gradvist.

En anden effekt, som man sikkert har set, er objekter, der hele tiden skifter farve. Effekten opnås ved at "invertere" objekt-dataene. "Invertere" vil sige at vende om på samtlige bits. Hvis en bit indeholder nul, sættes den til én og vice versa. Det giver en art filmnegativ.

REM invertieren
REM P.5.6-5

Første del svarer stort set til den første del af foregående program. Den første GET-sætning tjener til lettest muligt at finde de første tre værdier, som GET opbevarer ved gemningen af billedet. Derefter overgives objekt-dataene til fel-

tvariablen *Bild, som bringes på skærmen i PRESET-mode (altså inverteret) ved hjælp af PUT og gemmes straks med GET. Dernæst gemmes det inverterede objekt i en streng. Den begynder med en kopi af den første del af de oprindelige objektdata. Objektets egentlige billeddata sættes derimod med inverterede bits fra feltvariablen *Bild. For at kunne se resultatet, aktiveres til slut begge sprites og de ind- og udkobles efter tur i en WHILE-løkke.

5.7 Automatisk dimensionering af felter

Læseren har sikkert haft problemer med at skulle dimensionere feltvariabler, og kender to af fortolkerens fejlmeldinger. Subsrcipt out of range fremkommer, når en variabel er dimensioneret for lavt, Duplicate definition hvis man vil dimensionere den samme felt-variabel flere gange. I det følgende gennemføres en automatisk dimensionering, og det vises med et eksempel, at det beskytter mod begge typer fejl.

REM Fluesse
REM P.5.7-1

I den første programdel fastlægges indeksværdien for det første element i alle feltvariabler. Feltvariabler kan begynde med element nummer 0 eller 1.

> **OPTION BASE n**
> sætter indeksværdien for det første element i alle et programs feltvariabler til n (n kan kun være 0 eller 1)

Så dimensioneres en feltvariabel *fl$ for 10 elementer, og de indlæses fra DATA-sætninger. Nu skal indtastes et navn på en flod, der tildeles strengvariablen *flus$. Navnet søges i feltet *fl$, hvorunder UCASE-funktionen sørger for, at det ikke gør noget, hvis der er skrevet med store bogstaver. Er flodnavnet fundet, går det tilbage til start. Hvis ikke, kaldes rutinen *dimensionierung. I en FOR-løkke kopieres de enkelte feltelementer af variablen *fl$ over i variablen *fl1$, der tidligere er dimensioneret med det samme antal elementer, som *fl$ netop indeholder. Hvor mange elementer, *fl$ indeholder, fastslås med en Basic-funktion, der er specielt beregnet til det.

> **LBOUND (felt,dimension)**
> giver indeksnummeret på det første element af felt. Har feltet flere dimensioner, kan man angive med dimension, hvilke af dem, der skal undersøges

UBOUND (felt,dimension)
giver indeksnummeret på det sidste element i felt. Hvis feltet har flere dimensioner, kan man angive med dimension, hvilke af dem, der skal undersøges.

Et eksempel: Indtast følgende sætninger i direkte funktionsmåde.

```
OPTION BASE 1:DIM FL$(20) :PRINT LBOUND  (fl$) UBOUND (fl$)
```

Så får man tallene 1 (den nedre grænse)og 20 (den øvre grænse).

Men tilbage til programmet: Efter at det gamle felt er kopieret til det nye felt *fl1$, slettes det med ERASE og dimensioneres igen, men med et felt mere. Så kopieres elementerne tilbage fra *fl1$ til *fl$, og det sidste flodnavn tilføjes som sidste element. Endelig slettes *fl1$, og det går tilbage til starten.

5.8 Lidt matematik

Undertiden er det uundgåeligt at skulle foretage komplicerede beregninger og måske endda gentage dem flere steder. Det kan være besværligt at skulle indtaste lange, komplicerede formler flere gange. Så er det praktisk at kunne definere funktioner, og anvende dem som SIN og SQR.

DEF FNnavn (arg1,arg2,..) = funktionsdefinition
definerer en ny funktion, der kan kaldes på et andet sted med navn(..)

Navnene i parenteser bag funktionsnavnet er funktionens "argumenter", der kan benyttes ved beregningen af resultatet. (Funktionen SIN behøver f.eks. en vinkel som argument, hvortil sinus så beregnes.)

Et eksempel herpå: Antag at man flere gange skal beregne en retvinklet trekants areal. Den følgende sætning definerer funktionen *dreiecksflaeche, der gør det.

```
DEF FNtrekantsareal (grundl, hoej) = grundl*hoej/2
```

Indtast nu følgende sætninger i direkte funktionsmåde:

```
a=20:b=7:c=FNtrekantsareal (a,b): PRINT a, b, c
```

og man får

```
20  7  70
```

Følgende program viser et lidt større eksempel til anvendelse af selvdefinerede funktioner i Amiga-Basic

REM Kreisberechnung
REM P.5.8-1

I programmet defineres to funktioner til beregning af cirkelfladen med *DEF FNkrfl(r) og omkredsen med *DEF FNkrumf(r), og de anvendes senere, som om de er "fast indbygget" i Basic. Vi fortsætter med at kigge på, hvordan man kan opbygge et matematisk anvendelsesprogram.

REM Minimathe
REM

Der begyndes med at definere alle tal som dobbelt-præcisionstal og fastlægge værdien af tallet pi (?) i en dertil reserveret variabel. Derefter defineres omkredsfunktionen som vist. De efterfølgende menusætninger er kendte fra tidligere kapitler. Programmet sover så, som almindeligt under anvendelse af menuer, i en WHILE-løkke. I rutinen *menuKontrolle fastslås, hvilken menu brugeren har valgt.

> **v=MENU(n)**
> giver nummeret på den menutitel (n=0) eller det menupunkt (n=1), brugeren sidst valgte

Efter det er fastslået med MENU, hvad man ønsker at gøre, forgrener *menukontrolle til de enkelte subrutiner. De beskrives nu enkeltvis.

***wurzel:** Denne subrutine beregner kvadratroden af det indtastede tal ved hjælp af Basic-funktionen SQR. For at der ikke skal kunne opstå fejlmeldinger med SQR, sørges der for, at det indtastede tal er positivt.

> **v=ABS(x)**
> giver den absolutte værdi (uden fortegn) af det numeriske udtryk x

***expo:** Denne subrutine opløfter e (den naturlige logaritmes grundtal = 2.7182818284590) til det indtastede tals potens.

> **v=EXP(x)**
> giver potensen af x til grundtallet e (v=e^x)

***loga:** Subrutinen er modstykket til expo. Den beregner den naturlige logaritme af det indtastede tal ved hjælp af Basic-funktionen LOG.

v=LOG(x)
giver den naturlige logaritme af x til grundtallet e.

*kr: Denne subrutine giver beregning af omkreds og areal af cirklen med den indtastede radius. Den undersøger fortegnet på den indtastede værdi, før den bearbejdes. Hertil kan også anvendes en Basic-funktion.

v=SGN(x)
fortegnet af x overgives i form af en konstant

Værdi af x	SIGN-resultat
x<0	SGN(x) = -1
x=0	SGN(x) = 0
x>0	SGN(x) = 1

Tabel 5.7: Mulige resultater af SGN-funktionen

Er afprøvningen afsluttet, og resultatet ikke er negativt, beregnes cirkelfladen og omkredsen, og cirklen tegnes på skærmen.

*krausschn: Denne subrutine beregner arealet og buelængden af et lagkagestykke-formet cirkeludsnit med en selvdefineret funktion *FNbog, og tegner det på skærmen. (CIRCLE kan ikke kun tegne hele cirkler, men også udsnit deraf.) Det er nødvndigt med to indtastede værdier (radius og vinkel), som "hentes" af to forskellige subrutiner: *eingabe og *eingabe2.

*datum: Rutinen indeholder tre nye, endnu ikke anvendte kommandoer. Den udlæser den aktuelle dato i det øverste højre hjørne - og fungerer kun, hvis den korrekte dato er indstillet med "Preferences". Datoen gives - som sædvanlig - af en Basic-funktion.

v$=DATE$
giver en streng, der indeholder den aktuelle dato i formatet md-dg-år (altså f.eks.01-07-1989 for den 7. januar 1989)

Med to nye funktioner og én bekendt (LOCATE) sørger programmet for, at tekstcursoren står på samme sted i vinduet før og efter udlæsningen af datoen. Det sker ved at fastslå cursorens aktuelle position, lagre den, og genoprette den efter udlæsning af datoen.

v = POS(0)
giver den spalte, tekstcursoren netop befinder sig i

v = CRSLIN
giver den linie, tekstcursoren netop befinder sig på

Man kan ikke gøre så meget med dette lille matematik-program, men det kan udvides og forbedres efter behov. Prøv engang at indbygge følgende trigonometriske funktion og forsøg at vise dens resultater grafisk.

v = TAN(x)
giver tangens til vinklen x (der skal angives i radian)

5.9 Resten

Der er et par kommandoer, der ikke har været anvendt. For fuldstædnighedens skyld skal de kort beskrives.

DECLARE FUNCTION navn parameterliste LIBRARY
definerer en funktion, der realiseres af et maskinsprogs-underprogram, som befinder sig i en biblioteksfil. Funktionen anvendes som selvdefineret funktion efter at være defineret således.

LIBRARY"filnavn"
LIBRARY CLOSE
åbner et bibliotek for tilgang fra Amiga-Basic. Kommandoen er nødvendig før DEFINE FUNCTION-kommandoen for at meddele Basic, i hvilket bibliotek den deri definerede funktion, befinder sig

De to kommandoer udgør på elegant vis et bindeled mellem maskinsprog og Basic. Ethvert maskinsprogsbibliotek, der overholder en bestemt standard, kan anvendes sådan af Basic. Bibliotekerne har en indholdsfortegnelse med de enkelte rutiners navn, og hvor de findes. For at kunne tilgå et bibliotek fra Basic, skal indholdsfortegnelsen overføres til en såkaldt ".bmap-fil". Overføringen (konvertering) kan foretages med hjælpeprogrammet ConvertFD, der ligger i skuffen "BASICDemos". Kun når der foreligger sådan en .bmap-fil til et bibliotek, kan man udnytte rutinerne.
Der ligger to færdige .bmap-filer i skuffen "BASICDemos" (dos.library og graphics library). Et af dem benyttes i følgende eksempel. Hvis man er interesseret i at vide mere om dette tema, kan man finde det i Kapitel 6 og i Appendiks F af Basic-manualen; men man kommer ikke langt uden yderligere

håndbøger (feks. dem, der er udgivet af forlaget Addison-Wesley). Det er nemlig kun i disse håndbøger, man kan finde ud af, hvilke rutiner, der står til rådighed i hvilke biblioteker.

REM Biblio1
'P.5.9-1

(Programmet benytter .bmap-filen "graphics.library", der er i skuffen "BASIC-Demos".) Start programmet og tryk på venstre musetast. Fasthold tasten og bevæg musen. Det man ser på skærmen, vil næppe være muligt i Basic alene - i hvert fald ikke med den fart.

Endnu en Basic-funktion, der ikke har været omtalt tidligere, er LOC-funktionen

> **v = LOC(filnummer)**
> giver nummeret på datasættet i filen med direkte tilgang og betegnelsen filnummer, som sidst blev gået til med en GET- eller PUT-kommando

Et eksempel herpå: Indbygger man sætningen IF LOC(1)100 THEN pruefen bevirker den et hop til mærket *pruefen, hvis den sidste operation i en fil med direkte tilgang og betegnelsen nummer 1 har arbejdet med et datasæt bagved det 100. datasæt.

Den sidste ikke omtalte sætning er SHARED-sætningen indenfor underprogrammer, der særligt er af betydning ved programmering af store programmer med mange underprogrammer.

> **SHARED variabel1,variabel2,...**
> gør de anførte variabler til "globale variabler", der har samme værdi i underprogrammet og i resten af programmet

Hermed menes ikke, at sådanne variabler har en fast, uforanderlig værdi. Normalt er det sådan, at to variabler med samme navn i hovedprogrammet og i et underprogram er to forskellige objekter. Hvis variablen a i et underprogram tildeles værdien 4, og underprogrammet så afsluttes, giver PRINT a i hovedprogrammet ikke 4 af den grund. Istedet har variablen den værdi, der sidst blev tildelt variablen a i hovedprogrammet - også selv om tildelingen skete før tildelingen a=4 i underprogrammet. Med SHARED-sætningen kan man, om ønsket, undertrykke effekten. Ens navne i hoved- og underprogram svarer til ens objekter (samme hukommelsesadresser). To eksempler på det.

```
REM Eksempel 1
a=50:b=30
c=40:
CALL test
END

SUB test STATIC
PRINT a,b
END SUB
```

Programmet udlæser to gange nul på skærmen - selv om a og b har fået andre værdier i hovedprogrammet.

```
REM eksempel 2
a=50:b=30:c=40
CALL test
END

SUB test STATIC
SHARED a,b
PRINT a, b, c
END SUB
```

Det andet program giver derimod tallene 50, 30 og 0, fordi a og b nu betyder det samme i hoved- og underprogram. Variablen c er dog ikke defineret som SHARED og har ganske vist værdien 40 i hovedprogrammet, men nul i underprogrammet.

6 AFSKED

Nu blot en tak. Først til dem der har holdt ud til den bitre ende, og endelig til dem, der skabte Amigaen, for at de frembragte sådan et teknisk vidunder.

7 OVERSIGT OVER BASIC-KOMMANDOERNE

Program	kommandoer	Programnummer.
langsommere	GOSUB RETURN END	P1.4.2-1
tekstforgrening	ON GOTO	P1.4.2-2
plus	FOR NEXT	P1.4.3-1
trekant	FOR NEXT	P1.4.3-2
frigiv I	F-BLOCK	4.4-1
pil	WHILE WEND	P1.4.5-1
variabler	variabler	P1.5.2-1
postnum	endimensionalt felt	P1.5.3-1
trekantsareal	todimensionalt felt	P1.5.3-2
kasse	tredimensionalt felt	P1.5.3-3
og	AND	P1.6.2-1
eller	OR	P1.6.2-2
sammenligning	sammenligningsoperatorer	P1.6.3-1
linie	LINE GOTO	P2.1-1
tunnel	LINE, IF THEN, ELSE CLS	P.2.1-2
Billedskærm	PSET END	P2.2-1
billedskærm2	SCREEN WINDOW	P2.2-2
toSCREENS	FRE SPC TAB PTAB	
	WINDOW OUTPUT SCREEN CLOSE	P2.2-3
bæger	COS ATN SWAP WRITE	P2.3-1
farvetavle	PALETTE	P2.4-1
farvepalet	MERGE GOSUB RETURN	P2.4-2
regnbue	DEF type CHAIN MERGE COLOR	
	AREA WHILE WEND CIRCLE	
	AREAFILL PAINT	P2.5-1
afloesning	COMMON	P2.5-2
udfoer	CHAIN	P2.5-3
harer	CIRCLE	P2.5-4
skov	GET PUT DIM	P2.6-1
dam	ON MOUSE MOUSE ON SYSTEM CALL	
	SUB END SUB MOUSE-funktion	P2.6-2
tordenvejr	CLEAR INPUT INPUT$ PRINT# CVI	

	CVL CVS CVD OPEN LOCATE MKI MKS MKL MKD ERASE ON TIMER TIMER ON TIMER OFF	P2.7-1
mur	TIMER-funktion	P2.7-2
PAINTAmiga	ON MENU ON BREAK BREAK ON MENU ON SLEEP ON.GOSUB POINT FILES MENU MENU RESET BEEP UCASE$	P2.8-1
tegn	LPRINT LPOS	P3.1-1
Amiga-K	PAR:RESTORE READ DATA	P3.2-1
farve	READ DATA	P3.2-2
Amiga-L	grafik-print	P3.2-3
SW-hardcopy	WINDOW-funktion	P3.2-4
indud	PEEK POKE	P3.3-1
varlager	PEEKW PEEKL VARPTR	P3.3-2
feltlager	POKEL POKEW	P3.3-3
strenglager	SADD POKE	P3.3-4
ifflæsegemme	ASC FIX	P3.3-5
videofil	WRITE# CLOSE INPUT#	P3.4-1
kogefil	LINEINPUT LINE INPUT# CHR$ LEN STRING$ RIGHT$ MID$ LEFT$	P3.5-1
sorter	TIME$ VAL	P3.6-1
fejl	ON ERROR ERROR RESUME ERR ERL	P3.7-1
bilfil	SPACE$ PRINT USING	P3.8-1
dele	FIELD LSET PUT# GET#	P3.9-1
terminskalender	INSTR	P3.10-1
engelsk	TRANSLATE SAY	P4.1-1
egelsk2	TRANSLATE engelsk	P4.1-2
dansk	SAY-mode dansk	P4.1-3
dansk2	TRANSLATE dansk	P4.1-4
sprog	dansk sprogudlæsning	P4.1-5
regnemaskine	sproganvendelse	P4.1-6
toneskala	SOUND	P4.2-1
toneskala2	SOUND	P4.2-2
aftensang	musikeksempel	P4.2-3
folkevise	SOUND WAIT SOUND RESUME	P4.2-4
sinusbølge	WAVE	P4.3-1
trekantsbølge	WAVE	P4.3-2
savtandsbølge	WAVE	P4.3-3
pulsbølge	WAVE	P4.3-4
brusen	WAVE hvid brusen	P4.3-5
schwabisk vise	musikeksempel	P4.3-6

bilhorn	lydeffekt	P4.4-1
alarmtrin1	l ydeffekt	P4.4-2
brandvæsen	l ydeffekt	P4.4-3
mask.gevær	lydeffekt	P4.4-4
afklang	lydeffekt	P4.4-5
UFO-lander	lydeffekt	P4.4-6
motor	lydeffekt	P4.4-7
højspænding	lydeffekt	P4.4-8
laser	lydeffekt	P4.4-9

sprite-test	OBJECT.SHAPE OBJECT.VY OBJECT.X OBJECT.Y OBJECT.VX OBJECT ON	P4.5-1
bumletog	OBJECT.X-funktion OBJECT.STOP	P4.5-2
farvepragt	BOBs med 32 farver	P4.6-1
kunstflyvning	ON COLLISION COLLISION ON/OFF OBJECT.HIT OBJECT.PRIORITY	P4.6-2
joystick	STICK STRIG	P4.6-1
nedskydning	OBJECT.CLIP OBJECT.CLOSE OBJECT.AX OBJECT.AY COLLISION-funktion	P4.7-1
bevægelse	OBJECT.VY-funktion	P4.7-2
rulletest	SCROLL	P4.8-1
højhus	SCROLL	P4.8-2
softball	SCROLL	P4.8-3
fejlstart	PATTERN	P4.8-4
mønster	PATTERN	P4.8-5
indledning	SCROLL	P4.8-6
ramdisk	lagring af tekst på RAM-disk	P5.1-1
adventure	lagring af grafik på RAM-disk	P5.1-2
den store ABC	forstørrelse af bogstaver	P5.2-1
gæt steder	ASCII-koder	P5.3-1
lotto	RND	P5.4-1
terninger	RANDOMIZE TIMER	P5.4-2
GET-manipulation	GET	P5.5-1
fatamorgana	kipning af skærm	P5.5-2
objekt-data	læsning af sprites og BOBs	P5.6-1
sprite-farver	farveændring af sprites	P5.6-2
BOB-farver	OBJECT.PLANES	P5.6-3
opløs	opløsning af objekter	P5.6-4
invertering	invertering af objekter	P5.6-5

floder	LBOUND UBOUND OPTION BASE	P5.7-1
cirkelberegning	DEF FN	P5.8-1
minimat	MENU-funktion ABS EXP LOG	
	SGN DATE$ POS CRSLIN	P5.8-2
biblio	LIBRARY	P5.9-1

Oversigt over Basic kommandoer

Kommando	Kapitel
ABS	5.8
AND	1.6.2
AREA	2.5
AREAFILL	2.5
ASC	3.3
	2.7
ATN	2.3
BEEP	1.4.5
BREAK OFF	2.8.1
BREAK ON	2.8.1
BREAK STOP	2.8.1
CALL	2.6
CDBL	1.5.2
CHAIN	2.5
CHDIR	1.3.1
CHR$	3.5
CINT	1.5.2
CIRCLE	2.5
CLEAR	2.7
CLNG	1.5.2
CLOSE	3.4
CLS	1.4.2
COLLISON-funktion	4.6.1
COLLISON OFF	4.6
COLLISION ON	4.6
COLLISION STOP	4.6
COLOR	2.5
COMMON	2.5
CONT	1.2.5
COS	2.3
CSNG	1.5.2
CRSLIN	5.8
CVD	2.7
CVI	2.7
CVL	2.7
CVS	2.7

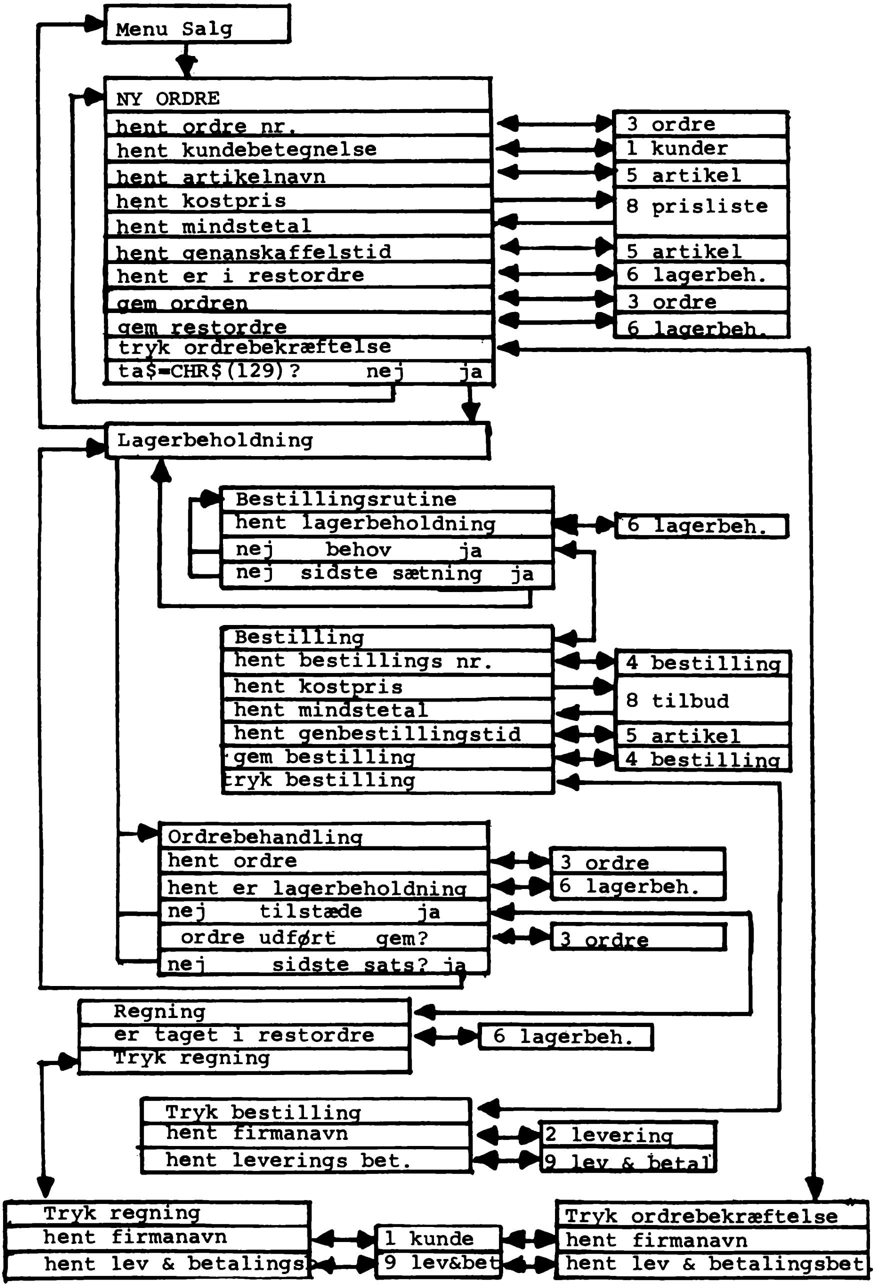
DATA	3.2
DATE$	5.8
DBL	1.5.2
DECLARE FUNCTION LIBRARY	5.9
DEF FN	5.8
DEF TYP	1.5.2
DELETE	1.2.4
DIM	1.5.3
END	1.4.2
END SUB	2.6
EOF	2.7
ERASE	2.7
	3.4
ERL	3.7
ERR	3.7
ERROR	3.7
EXIT SUB	2.6
EXP	5.8
FIELD	3.9
FILES	1.3.1
FIX	3.3
FOR...NEXT	1.4.3
FRE	2.2
GET til grafik	2.6
GET til filer	3.9
GOSUB	1.4.2
GOTO	1.4.2
HEX$	4.8
IF...THEN	1.4.4
INKEY$	2.2
INPUT	1.5.3
INPUT$	2.7
INPUT#	3.4
INSTR	3.10
INT	1.5.2
KILL	1.3.7
LBOUND	5.7
LEFT$	3.5
LEN	3.5
LET	1.5.2
LIBRARY	5.9
LIBRARY CLOSE	5.9

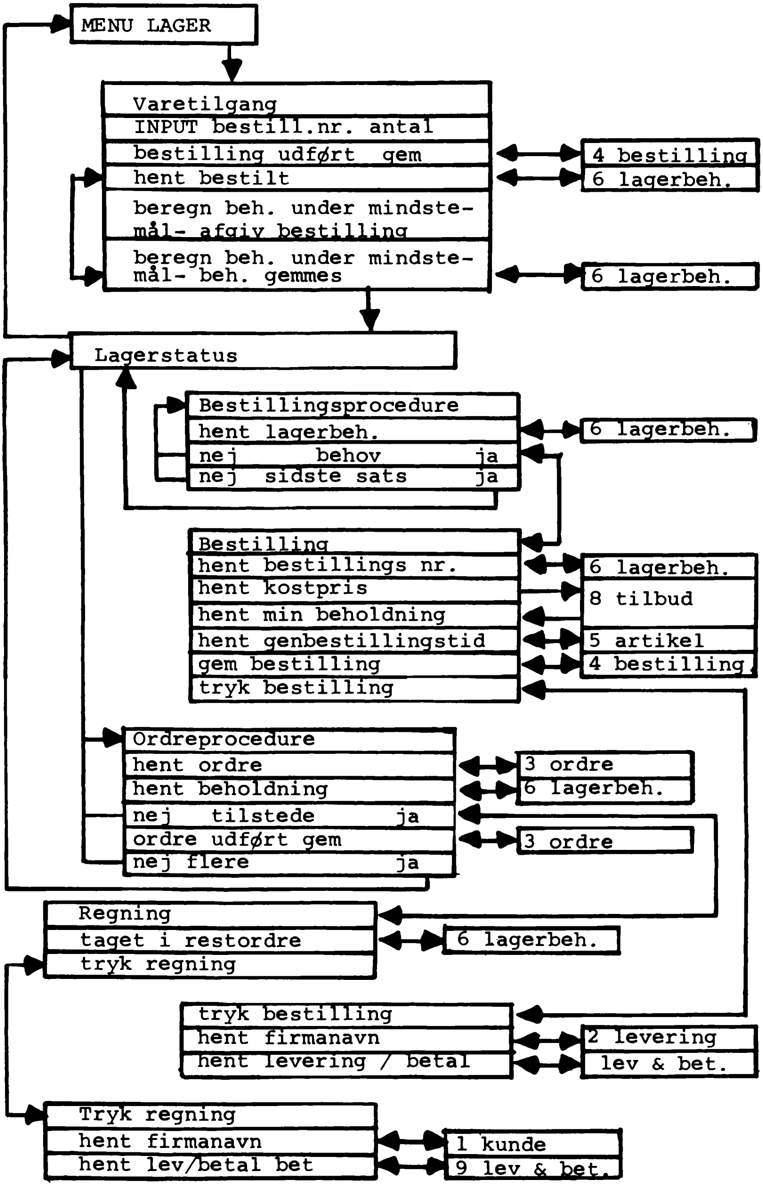
LINE	2.1
LINE INPUT	3.5
LINE INPUT#	3.5
LIST	1.2.4
LLIST	3.1
LNG	1.5.2
LOAD	1.3.3
LOC	5.9
LOCATE	1.4.3
LOF	5.6
LOG	5.8
LPOS	3.1
LPRINT	3.1
LPRINT USING	3.1
LSET	3.9
MENU	2.8.1
MENU-funktion	5.8
MENU OFF	2.8.1
MENU ON	2.8.1
MENU STOP	2.8.1
MENU RESET	2.8.1
MERGE	2.4
MID$	3.5
MKD$	2.7
MKI$	2.7
MKL$	2.7
MKS$	2.7
MOUSE-funktion... 2.8	2.6
MOUSE OFF	2.6
MOUSE ON	2.6
MOUSE STOP	2.6
NAME	1.3.7
NEW	1.3.2
OBJECT.AX	4.7
OBJECT.AY	4.7
OBJECT.CLIP	4.7
OBJECT.CLOSE	4.7
OBJECT.HIT	4.6
OBJECT.OFF	4.5
OBJECT.ON	4.5
OBJECT.PLANES	4.6

OBJECT.PRIORITY	4.6
OBJECT.SHAPE	4.5
OBJECT.START	4.5
OBJECT.STOP	4.5
OBJECT.VX	4.5
OBJECT.VX-funktion	4.7
OBJECT.VY	4.5
OBJECT.VY-funktion	4.7
OBJECT.X	4.5
OBJECT.X-funktion	4.5
OBJECT.Y	4.5
OBJECT.Y-funktion	4.5
OCT$	4.8
ON.GOSUB	1.4.2
ON GOTO	1.4.2
ON BREAK	2.8.1
ON COLLISION	4.6
ON ERROR	3.7
ON MENU	2.8.1
ON MOUSE	2.6
ON TIMER	2.7
OPEN	2.7
	3.4
	3.5
OPTION BASE	5.7
PAINT	2.5
PALETTE	2.4
PATTERN	4.8
PEEK	3.3
PEEKL	3.3
PEEKW	3.3
POINT	2.8.1
POKE	3.3
POKEL	3.3
POKEW	3.3
POS	5.8
PRESET	2.2
PRINT	1.2.2
PRINT#	2.7
	3.4
PRINT USING	3.8
PSET	2.2

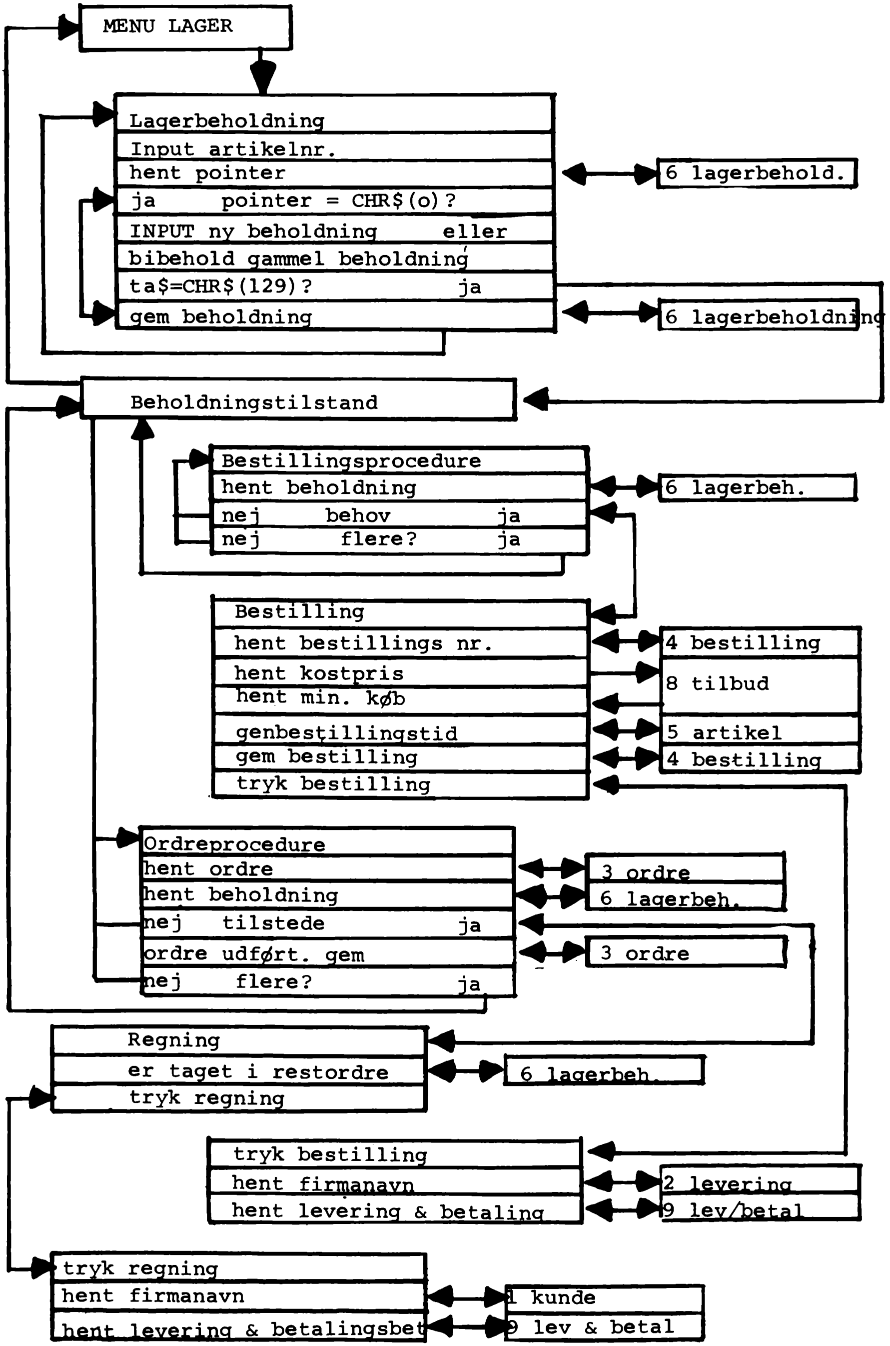
PTAB	2.2
PUT til grafik	2.6
PUT til filer	3.9
RANDOMIZE	5.4
RANDOMIZE TIMER	5.4
READ	3.2
REM	1.4.2
RESTORE	3.2
RESUME	3.7
RETURN	1.4.2
RIGHT$	3.5
RND	4.4
RSET	3.9
RUN	1.2.5
SADD	3.3
SAVE	1.3.5
SAY	4.1
SCREEN	2.2
SCREEN CLOSE	2.2
SCROLL	4.8
SGN	5.8
SHARED	5.9
SIN	4.3
SLEEP	1.4.6
SNG	1.5.2
SOUND	4.2
SOUND RESUME	4.2
SOUND WAIT	4.2
SPACE$	3.8
SPC	2.2
SQR	4.8
STICK	4.6.1
STOP	1.2.5
STR	1.5.2
STRIG	4.6.1
STRING$	3.5
SUB	2.6
SWAP	2.3
SYSTEM	1.3.6
TAB	2.2
TAN	5.8
TIME$	3.6

TIMER-funktion	2.7
TIMER OFF	2.7
TIMER ON	2.7
TIMER STOP	2.7
TRANSLATE$	4.1
TROFF	1.2.5
TRON	1.2.5
UBOUND	5.7
UCASE$	2.8.1
VAL	3.6
VARPTR	3.3
WAVE	4.3
WHILE-WEND...	1.4.5
WIDTH	3.1
WINDOW	2.2
WINDOW CLOSE	2.2
	2.8.1
WINDOW-funktion	2.2
WINDOW OUTPUT	2.2
	2.8.1
WRITE	2.3
WRITE#	3.4
XOR	2.6


Menu Salg
NY ORDRE
hent ordre nr.
hent kundebetegnelse
hent artikelnavn
hent kostpris
hent mindstetal
hent genanskaffelstid
hent er i restordre
gem ordren
gem restordre
tryk ordrebekræftelse
ta$=CHR$(129)? nej ja
3 ordre
1 kunder
5 artikel
8 prisliste
5 artikel
6 lagerbeh.
3 ordre
6 lagerbeh.
Lagerbeholdning
Bestillingsrutine
hent lagerbeholdning
nej behov ja
nej sidste sætning ja
6 lagerbeh.
Bestilling
hent bestillings nr.
hent kostpris
hent mindstetal
hent genbestillingstid
gem bestilling
tryk bestilling
4 bestilling
8 tilbud
5 artikel
4 bestilling
Ordrebehandling
hent ordre
hent er lagerbeholdning
nej tilstæde ja
ordre udført gem?
nej sidste sats? ja
3 ordre
6 lagerbeh.
3 ordre
Regning
er taget i restordre
Tryk regning
6 lagerbeh.
Tryk bestilling
hent firmanavn
hent leverings bet.
2 levering
9 lev & betal
Tryk regning
hent firmanavn
hent lev & betalingsb
1 kunde
9 lev&bet
Tryk ordrebekræftelse
hent firmanavn
hent lev & betalingsbet.



MENU LAGER
Varetilgang
INPUT bestill.nr. antal
bestilling udført gem
hent bestilt
beregn beh. under mindste-mål- afgiv bestilling
beregn beh. under mindste-mål- beh. gemmes
4 bestilling
6 lagerbeh.
6 lagerbeh.
Lagerstatus
Bestillingsprocedure
hent lagerbeh.
nej behov ja
nej sidste sats ja
6 lagerbeh.
Bestilling
hent bestillings nr.
hent kostpris
hent min beholdning
hent genbestillingstid
gem bestilling
tryk bestilling
6 lagerbeh.
8 tilbud
5 artikel
4 bestilling
Ordreprocedure
hent ordre
hent beholdning
nej tilstede ja
ordre udført gem
nej flere ja
3 ordre
6 lagerbeh.
3 ordre
Regning
taget i restordre
tryk regning
6 lagerbeh.
tryk bestilling
hent firmanavn
hent levering / betal
2 levering
lev & bet.
Tryk regning
hent firmanavn
hent lev/betal bet
1 kunde
9 lev & bet.



MENU LAGER
Lagerbeholdning
Input artikelnr.
hent pointer
ja pointer = CHR$(o)?
INPUT ny beholdning eller
bibehold gammel beholdning
ta$=CHR$(129)? ja
gem beholdning
6 lagerbehold.
6 lagerbeholdning
Beholdningstilstand
Bestillingsprocedure
hent beholdning
nej behov ja
nej flere? ja
6 lagerbeh.
Bestilling
hent bestillings nr.
hent kostpris
hent min. køb
genbestillingstid
gem bestilling
tryk bestilling
4 bestilling
8 tilbud
5 artikel
4 bestilling
Ordreprocedure
hent ordre
hent beholdning
nej tilstede ja
ordre udført. gem
nej flere? ja
3 ordre
6 lagerbeh.
3 ordre
Regning
er taget i restordre
tryk regning
6 lagerbeh.
tryk bestilling
hent firmanavn
hent levering & betaling
2 levering
9 lev/betal
tryk regning
hent firmanavn
hent levering & betalingsbet
1 kunde
9 lev & betal

OMEGA STEREO SAMPLER
VERDENS BEDSTE
TROR VI SELV - MEN PRØV DEN!
OG BLIV OVERBEVIST
• Lav dit eget studio.
• Lav perfekte lydmix og musikstykker.
• Brug det hele i dine egne spil.
• Optag et nys, et host eller naboens hund.
• Lav et trommesæt med lydene.
Køb »TESTVINDEREN I COMPUTER«.
Årets hit i USA.
Det er dansk, når det er bedst..!
Fa. DANTECH 8840 Rødkærsbro.
OMEGA STEREO SAMPLER
STEREO SAMPLER
FOR
AMIGA 500/2000
OMEGA STEREO SAMPLER
Grossist
NCS
Postboks 105, Smedegade 7
DK-6950 Ringkøbing. Tlf. 07 32 02 02

AMIGA HARDWARE

Hvad kan jeg få af tilbehør til min Amiga?
Hvad kan det?
Hvordan bruges det? og til hvad?

Med denne bog har du et godt kort på hånden, inden du går ud og køber ind.

Du kan spare hundrede, ja tusinder af kroner, når du har undersøgt markedet til bunds, med denne bog.

KUN KR. 99,-

AMIGA DOS CLI

CLI (Command line interface) beskrives grundigt i denne bog.
AMIGA DOS kan anvendes på to måder, nemlig ved brug af mus, og som bogen beskriver CLI.
Alle kommandoer med syntax og parametre vises med tilhørende eksempler.
Denne bog er så god, at Commodore Data Danmark har taget den til sig som »Dansk DOS Manual«.

KUN KR. 199,-

Programmør/Forfatter søges

Har Du skrevet/udviklet gode programmer til AMIGA, var det måske en ide at tage kontakt til Forlaget N.C.S.
Vi er altid på jagt efter nyt og spændende kvalitets-software, for salg i hele Scandinavien.
Mener Du, at netop Dit program har interesse for mange andre end Dig selv, så skriv et par ord til:

NORDIC COMPUTER SOFTWARE
Smedegade 7
DK-6950 Ringkøbing

EN DATA BECKER BOG
WELTNER & HORNIG
COMMODORE
AMIGA
TIPS OG TRICKS
En guldgrube for AMIGA-folk
DANSK
NORSK
udgave
Forlag: NCS, box 105, DK 6950 Ringkøbing

248,-

EN DATA BECKER BOG
DITTRICH
COMMODORE
AMIGA
MASKINSPROG
Maskinsprog gør livet lettere!
DANSK
udgave
NORSK
Forlag: NCS, box 105, DK 6950 Ringkøbing

248,-

EN DATA BECKER BOG
SCHUMANN
COMMODORE
AMIGA
& ATARI m. fl.
C for begyndere
Sproget C er AMIGA-venligt
DANSK
NORSK
udgave
Forlag: NCS, box 105, DK 6950 Ringkøbing

248,-

SPECIALFORLAG FOR COMPUTERBØGER
L. J. HELBO
AMIGA
GRAFIK & MUSIK
GENVEJEN TIL KREATIVITET!
DANSK
Forlag: , box 105, DK-6950 Ringkøbing

298,-

TIL NOTATER:

TIL NOTATER:

TIL NOTATER:

TIL NOTATER:

M2 AMIGA

Modula-2 COMPILER

Det superhurtige udviklingssystem

nu hos NCS

NORDIC COMPUTER SOFTWARE
SMEDEGADE 7 · DK 6950 RINGKØBING

Kundeservice:
Tlf. 07 32 02 02 kl. 9.00-10.00

Fra udlandet

Vi er normalt lagerførende i nedenstående:

Addison-Wesley:

Intuition Reference Manual
Rom Kernal Reference Manual: Exec
Rom Kernal Reference Manual: Libraries and Devices
Hardware Reference Manual

Abacus:

Amiga System Guide
Amiga Disk Drive

Sybex:

Amiga Haandbook VOL I
Amiga Haandbook VOL II
Amiga Programmers guide
Programming The 68000

Tewi:

The M68000 Family VOL I

Markt&Technik

ISBN 87-7283-030-1

This was brought to you

from the archives of

http://retro-commodore.eu